Escuela
Politécnica
Superior

Uso de SVG para desarrollar mapas web para personas con discapacidad visual

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Sergio Juan Armero
Tutor/es:
Sergio Luján Mora

Universitat d'Alacant
Universidad de Alicante

Preámbulo

Este trabajo enfrenta el problema de la accesibilidad en mapas web geográficos. Un usuario con discapacidad no puede utilizar, hoy en día, un mapa web con las mismas funciones que otros usuarios utilizan. Este problema significa un gran agujero en la usabilidad universal, ya que una gran cantidad de personas se ven excluidas. Esto contraviene claramente los principios de la Web (World Wide Web Consortium, 1997), que fue creada para el beneficio de cualquiera en cualquier circunstancia. Además, esta clase de mapas podría ser muy útil para personas con discapacidad, pero estas no pueden aprovecharlos debido a los problemas de accesibilidad. En este proyecto, algunas de las últimas tecnologías como, por ejemplo, PostGIS o SVG, son utilizadas para proporcionar características de accesibilidad en un mapa web geográfico. Como no hay una forma estándar de hacer esto, se investiga, diseña e implementa un sistema que conste de los componentes esenciales como para proveer características de accesibilidad. Lo que proponemos es una de las posibles soluciones, con sus ventajas y limitaciones, al problema de la accesibilidad en mapas web geográficos. La solución propuesta abre una discusión sobre el resultado obtenido y algunos comentarios sobre el estado actual de las tecnologías involucradas en la accesibilidad web.

Para abordar este tema, se han adquirido una serie de conocimientos antes y durante la ejecución de la parte práctica del proyecto. En concreto, se han investigado áreas relacionadas con el tema y las tecnologías del sistema a plantear.

Accesibilidad web Se han aprendido conceptos básicos sobre accesibilidad web en el curso creado por Sergio Luján Mora, Ester Serna Berná, Olga Carreras, Armando Suárez Cueto, Antonio Ferrández Rodríguez y la Cátedra Telefónica Universidad de Alicante[1]. Este curso cubre la detección de problemas de accesibilidad en páginas web, las soluciones para los problemas más comunes de accesibilidad web, el desarrollo de páginas accesibles y la mejora del posicionamiento de las mismas en buscadores. El curso fue de gran utilidad para introducir una visión de los problemas que enfrentan los usuarios discapacitados a la hora de navegar, así como de las tecnologías que existen hoy en día para ayudarles. Además de este curso, se exploran otros recursos que tratan la accesibilidad web, como pueden ser artículos científicos, entradas en blogs especializados,

[1] https://www.udemy.com/aprende-accesibilidad-web-paso-a-paso/

tutoriales sobre tecnologías web o artículos de revistas.

PostgreSQL y PostGIS Aunque durante el Grado de Ingeniería Informática se estudian los fundamentos, el análisis y el diseño de bases de datos, la parte práctica se centra en bases de datos con MySQL como motor. Para este trabajo, se decidió utilizar PostgreSQL, ya que su extensión PostGIS permite la gestión de información geográfica de forma eficiente, así como consultas y operaciones sobre ella. Se estudió la instalación y el funcionamiento de este motor y esta extensión mediante la documentación oficial, páginas especializadas y páginas de consultas sobre informática.

QGIS Se aprendió el uso básico de este editor de datos geográficos para poder importar la información disponible en Sistema de Información Geográfica de la Universidad de Alicante (SIGUA), crear otras capas basadas en ella y exportarlas en el formato y proyección adecuados. Se investigó también, por lo tanto, sobre los formatos disponibles para representar geografía y las proyecciones en las que se puede hacer.

Otra herramienta (shp2pgsql-gui) complementa a QGIS, ya que permite incorporar los datos creados en él a la base de datos. También se aprendió a hacer un uso básico de ella.

Gráficos vectoriales Un buen conocimiento de los gráficos vectoriales es primordial para plantear el proyecto. Para este trabajo, se ha aprendido sobre el funcionamiento de SVG, su forma de representar formas, y sus especificaciones publicadas. Se hizo énfasis en la forma en la que SVG puede ayudar en la accesibilidad de mapas web.

Escritura de textos científicos Unas de las actividades que involucra este proyecto es la redacción y presentación de un artículo científico en inglés, aparte de esta memoria, a la revista "Enfoque UTE"[2]. Esto requiere aprender la estructura básica de un texto de estas características (conocida como IMRaD). También sirve para conocer el proceso de revisión, corrección y presentación de textos científicos, así como los estilos y formatos que se exigen. La confección del artículo resultó ser muy útil para ganar experiencia en la redacción, síntesis y exposición de ideas.

[2]http://ingenieria.ute.edu.ec/enfoqueute/index.php/revista

Agradecimientos

Me gustaría dar las gracias a mi tutor, Sergio Luján Mora, por toda su ayuda para desarrollar este trabajo y por la oportunidad que me dio de estudiar un campo tan interesante como necesario. Además, tanto mi tutor como yo queremos agradecer al SIGUA su colaboración con los datos geográficos y, en especial, a José Manuel Mira por sus buenos consejos y opiniones sobre el proyecto. Por último, pero no menos importante, quisiera agradecer a mi familia y amigos todo su apoyo durante estos años. Gracias a todos por haberme hecho llegar hasta aquí.

A Cris.

"Demasiado ego para polvo de planeta"

— Ana Isabel García Llorente

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

1 Introducción

Las personas con discapacidad suponen entre el 15% y el 20% de la población mundial mayor de 15 años (World Health Organization, 2011). La accesibilidad consiste en qué productos, sistemas, entornos o instalaciones pueden ser utilizadas por personas con el rango más amplio de capacidades (International Organization for Standardization, 2014). Aunque las instituciones de todo el mundo son cada vez más conscientes de la necesidad de la accesibilidad en infraestructuras, las personas con discapacidad siguen sufriendo dificultades hoy en día. Un ejemplo de esta concienciación podría ser la iniciativa "Córdoba accesible" (Instituto Municipal de Desarrollo Económico y Empleo de Córdoba, 2013), que pretende recoger información sobre la accesibilidad de la ciudad de Córdoba, España. Otro ejemplo podría ser la implementación de un sistema de accesibilidad para mapas físicos en Getxo y en Madrid (PUNTODIS, 2017). Además, la Fundación Telefónica promociona una aplicación que permite a los usuarios recopilar información sobre problemas de accesibilidad en sus alrededores, para que otros puedan acceder a esa información y tenerla en cuenta (Fundación Telefónica, 2018).

Las personas con discapacidad usan varias herramientas para llevar a cabo tareas cotidianas. Estas son, entre otras, perros guía o bastones. Ellas también usan sus otros sentidos para comprender su entorno. En este aspecto, los pavimentos táctiles son muy relevantes. Según la forma de su relieve, los pavimentos podotáctiles tienen un significado diferente para una persona ciega. Los caminos se representan con líneas, mientras que los círculos se utilizan para alertar (Donoso, 2018). Sin embargo, su uso debe ser bien planeado, porque un mal diseño puede llevar a la gente ciega a malentendidos.

Algunos dispositivos de ayuda funcionan también como protección. Por ejemplo, el color de los bastones se utiliza normalmente para identificar la discapacidad que una persona tiene. En los Estados Unidos, un bastón totalmente blanco pertenece a una persona totalmente ciega, mientras que un bastón blanco con líneas rojas es usado normalmente por personas que aún preservan algo de visión (Lewis, 2018). Esto puede hacer que, por ejemplo, los conductores conozcan la discapacidad específica de una persona y actúen en consecuencia.

Es cierto que se han tomado pasos muy grandes hacia la accesibilidad en el "mundo físico", esto es, infraestructuras, edificios, carreteras y transporte público (entre otros). También existen organizaciones que trabajan para proveer a usuarios con discapacidades con trabajo y cierta independencia económica.

Sin embargo, este progreso no ha tenido lugar en el "mundo digital" (Ellis & Kent, 2017). Si nos concentramos en las nuevas tecnologías, las personas con discapacidad también encuentran dificultades en su uso. Esto se debe a que estas tecnologías aún no están adaptadas a este tipo de usuarios, como los edificios o caminos no lo estaban hace un tiempo. Como los ordenadores, móviles y otras tecnologías son cada vez más populares (y, por supuesto, más útiles), es necesario avanzar en el campo de la accesibilidad web.

Podemos decir que un sitio web es accesible si su efectividad, utilidad, usabilidad y eficiencia es la misma para usuarios discapacitados como usuarios no discapacitados (Brajnik, 2009). Cuando se quiere desarrollar un sitio web accesible, debemos tener en cuenta que existen distintas discapacidades que pueden afectar de forma distinta a la experiencia de usuario (World Health Organization, 2001).

Visual Las personas con ninguna o poca visión tienen problemas para percibir los colores o ver los elementos a su alrededor. Por ejemplo, una persona daltónica tiene problemas distinguiendo las líneas rojas y verdes del metro cuando se cruzan.

Auditiva Las personas con este tipo de discapacidad tienen problemas percibiendo sonidos. Por ejemplo, una persona sorda puede tener problemas al asistir a una clase presencial o conferencia, ya que no suelen contar con traducción a la lengua de signos o subtítulos.

Física Las personas con discapacidad física sufren un decremento de sus habilidades motoras. Por ejemplo, una persona en silla de ruedas puede ver su movilidad afectada cuando pretende acceder a edificios sin rampas o ayudas para este tipo de discapacidad.

Intelectual Las personas con discapacidad intelectual tienen problemas con procesos como la lectura de textos o con la comprensión de conceptos y del espacio que les rodea. Por ejemplo, una persona que sufra de discalculia (Butterworth, Varma & Laurillard, 2011) no puede realizar directamente operaciones matemáticas sencillas o utilizar un mapa de forma efectiva, porque se desorienta con facilidad.

Sin embargo, es cierto que no todas las discapacidades afectan al uso de la Web o de las nuevas tecnologías (Pressbooks Ryerson University, 2019). Por ejemplo, una persona en silla de ruedas puede, en principio, utilizar la Web de la misma forma que una persona sin discapacidad lo hace. Su falta de movilidad no afecta a la manera en la que se usa esa tecnología. Otras discapacidades sí suponen un cambio en cómo se afronta el uso de la Web, sobre todo aquellas relacionadas con la percepción y el procesamiento de información. A continuación se indaga en cada una de ellas.

En primer lugar, encontramos la pérdida total de la visión. Esta discapacidad afecta profundamente el uso de la Web, debido a la naturaleza visual de la misma. Suelen ayudarse de un lector de pantalla,

que es un programa que mediante un sintetizador de voz describe al usuario los elementos que se encuentran en la pantalla, ayudándole a navegar por ella. La mayoría de las veces, las personas ciegas tendrán problemas:

- Con la navegación de la página, ya sea por redirecciones inesperadas, por una navegación inconsistente o porque directamente se hace imposible navegar al contenido.
- Con los controles de la página, bien porque no puedan operarse por teclado o bien porque realicen acciones inesperadas (que ocurran, por ejemplo, sin su activación por parte del usuario).
- Con el propio contenido de la página, que puede resultar muy complejo para ser navegado sin utilizar la visión, o que no tiene alternativa textual si se trata de material gráfico.

En segundo lugar, hablaremos de la pérdida parcial de la visión. Esta discapacidad causa que, aunque los elementos del sitio web se perciban, se haga de forma que necesitan un programa de magnificación para poder verlos bien. Un programa de magnificación es un software que permite al usuario agrandar ciertas zonas de la pantalla para poder percibirlas mejor. Aunque algunas lo hacen, las personas con poca visión no suelen utilizar lectores de pantalla. En cuanto al uso de la Web, estas personas encuentran algunos problemas distintos a las personas ciegas, que son:

- El poco contraste o poca claridad de los textos al aumentarlos. Por ejemplo, una mala combinación de colores (o su falta de personalización) puede causar que el usuario no perciba bien los textos. Otro problema que surge es la pérdida de calidad de los textos insertados en imágenes cuando éstos se magnifican mediante software.
- El contenido con tamaños fijos. Esto hace que no se adapten correctamente a los cambios de *zoom* que los usuarios con poca visión pueden aplicar para verlos mejor, haciendo que sean de difícil percepción para ellos.
- Como en el caso anterior, la navegación problemática o las acciones inesperadas pueden llevar también a problemas de accesibilidad en personas de baja visión.

En tercer lugar, encontramos a las personas con sordera parcial o total. Estas personas tienen realmente pocos impedimentos a la hora de utilizar la Web ya que, como hemos dicho antes, su esencia es mayoritariamente visual. Sin embargo, encuentran dificultades en:

- Contenidos multimedia que no tengan subtítulos o una transcripción de los mismos.
- Contenidos multimedia en vídeo que no tengan interpretación en lengua de signos.

En cuarto lugar, encontramos a personas con discapacidades físicas en extremidades superiores, que les impide utilizar un ratón correctamente. Estas personas suelen apoyarse en control por voz o en un teclado. Los problemas a los que se enfrentan tienen que ver con los controles del sitio web, en concreto:

- Los controles son muy pequeños, lo que hace que no pueda hacerse clic en ellos con facilidad.
- Elementos de control que no pueden ser operados mediante el teclado.

En quinto y último lugar, podemos hablar de personas con discapacidad cognitiva. Esta discapacidad contiene un alto rango de grados, en el que puede haber personas que simplemente tengan dificultades de lectura, hasta personas con problemas más severos en el lenguaje y el procesamiento de ideas. Sin embargo, hay una serie de problemas comunes a todas ellas:

- Contenido demasiado abundante o mal estructurado, que causa que no comprendan bien el sitio web en general, lo que afecta también a la navegación por el mismo.
- El lenguaje que se utiliza en el contenido es demasiado complejo o avanzado. Esto causa que no se comprenda correctamente el contenido concreto de la página web, y puede llevar a una situación de frustración del usuario.
- Una navegación complicada o poco clara, que causa que no puedan desenvolverse correctamente por el sitio.

Como se ha mencionado anteriormente, las personas con discapacidad utilizan herramientas como bastones para llevar a cabo tareas cotidianas en el "mundo físico". Estas personas cuentan también con herramientas para ayudarles a usar las nuevas tecnologías en el "mundo digital". Múltiples productos de apoyo (PA) existen para hacer el uso de las nuevas tecnologías más fácil para personas con discapacidad, como por ejemplo, programas reconocimiento de voz, aplicaciones de ampliación de la pantalla o lectores de pantalla. Estos últimos son extremadamente importantes para personas con discapacidad visual, y requieren tanto una estructura de la página como unos atributos en los elementos específicos para funcionar correctamente. Ahora mismo, la implementación de los lectores de pantalla más utilizados no solo difiere entre ellos sino que, aún peor, no soporta la especificación de Accessible Rich Internet Applications (ARIA) completamente (PowerMapper, 2018).

La especificación ARIA (World Wide Web Consortium, 2017) provee diversas formas de mejorar la accesibilidad de un sitio web. En particular, dos características principales están disponibles: roles y atributos. Los primeros definen o redefinen la semántica de los elementos, siempre que el cambio no

entre en conflicto severo con las semánticas nativas. Los segundos son una forma potente de proveer información suficiente sobre un elemento para que un lector de pantalla pueda enunciarlo correctamente. Por ejemplo, se pueden indicar los títulos, las descripciones o si un elemento debe ser visible a lectores de pantalla.

Otros PA son las líneas *braille* que, usando mecanismos electromagnéticos, representan caracteres *braille* según sea necesario. Las personas ciegas navegan la web con estos dispositivos. Para los usuarios físicamente discapacitados existen los "apuntadores", que les permiten apuntar a la pantalla usando solo sus cabezas, así que aunque está pensado para personas con problemas motores, deben preservar un buen control de su cabeza para utilizarlos (Luján-Mora, 2009).

Para desarrollar sitios web accesibles, algunas directrices deben ser seguidas. Específicamente, el World Wide Web Consortium (W3C) desarrolla una serie de directrices conocidas como Web Content Accessibility Guidelines (WCAG) con el objetivo de proporcionar un estándar compartido internacional. Estas directrices definen tres niveles de accesibilidad (A, AA, AAA) siendo el último el mejor de ellos. Un sitio web recibe un nivel dependiendo de las directrices que cumpla y la prioridad de ésta. Por ejemplo, cumplir con las directrices de prioridad uno, dos y tres proporciona un nivel triple-A al sitio web.

Este proyecto no intenta cubrir la accesibilidad web en su totalidad, ya que es un tema muy amplio. Nos centraremos en mapas web geográficos (también llamados mapas en línea geográficos), es decir, mapas disponibles en línea (como una página web) que presentan o describen la geografía de una cierta zona. Un mapa geográfico puede representar tanto interiores como exteriores. En nuestro caso, la solución propuesta recoge información geográfica sobre los exteriores del campus de la Universidad de Alicante. Nuestro sistema está diseñado para funcionar con mapas exteriores, pero podría ser utilizado en mapas interiores, aplicando las correspondientes adaptaciones.

Este trabajo pretende construir un sistema que provea características de accesibilidad a mapas web geográficos, para tratar de resolver un problema al que usuarios con discapacidades se enfrentan: no pueden aprovechar los beneficios que los mapas en línea pueden aportarles (independencia y autosuficiencia). El resultado es una aplicación web con un diseño *responsive*, que funciona en ordenadores y en móviles, aprovechando los sensores de estos últimos (por ejemplo, GPS, brújula o giroscopio). El resultado se clasifica como un Mapa Digital Interactivo (Ducasse, Brock & Jouffrais, 2017), porque se proyecta en una pantalla y se interactúa con él de forma indirecta mediante un teclado, un ratón u otras herramientas o dispositivos.

De todas las formas en las que podríamos representar un mapa con tecnologías web, podemos destacar tres: *tiles*, *canvas* y Scalable Vector Graphics (SVG). Los *tiles* suelen utilizar imágenes rasterizadas, esto es, formatos que representan las imágenes píxel a píxel, especificando el color de cada uno en ma-

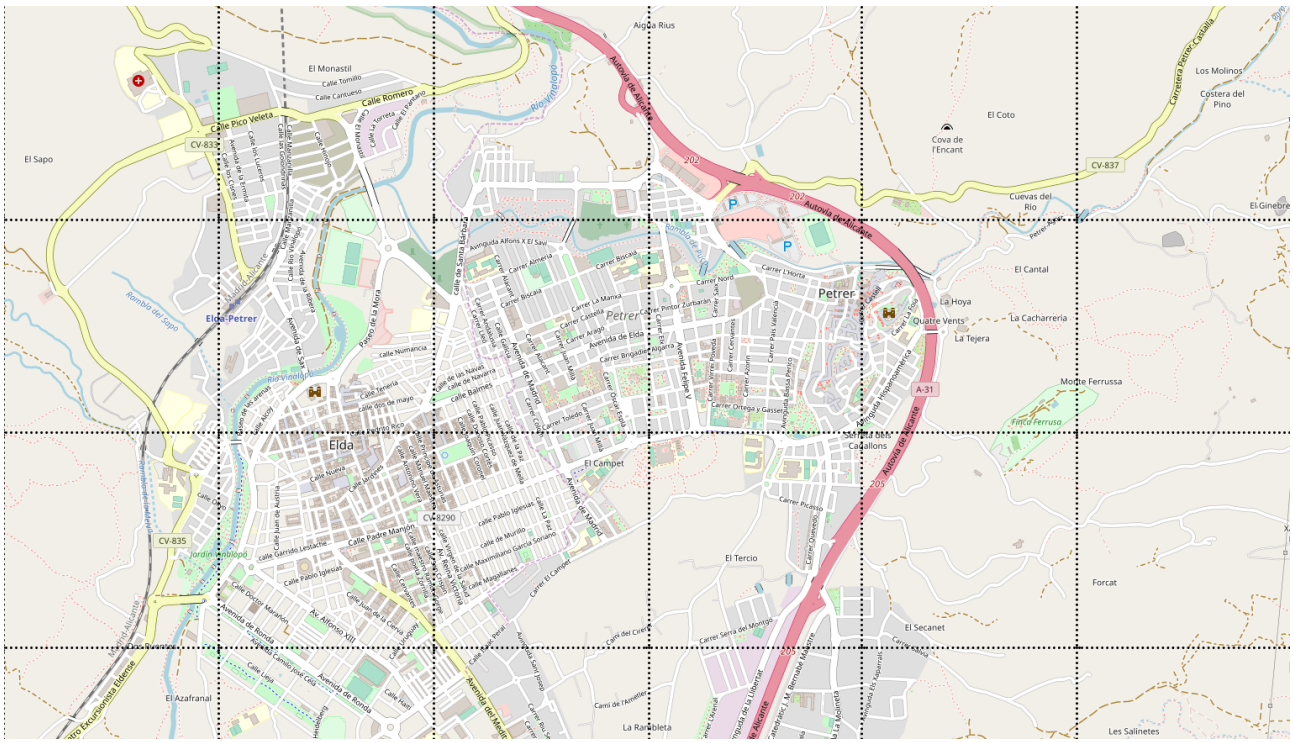


Figura 1.1: Tiles (divididos por líneas negras punteadas) que conforman el mapa de dos poblaciones alicantinas

trices rectangulares (Microsoft, 2017). Esto significa que los mapas se representan juntando varias imágenes en la posición correcta, como si se tratara de un puzle. Esto puede encontrarse visualmente en la Figura 1.1. Cuando la imagen se escala, pierde calidad. Además, la representación píxel a píxel impide incluir información de accesibilidad porque no se pueden identificar conjuntos significativos como edificios, caminos...

Por otro lado, los *canvas* se utilizan para dibujar gráficos de forma programática. En concreto, se utiliza la Canvas API (World Wide Web Consortium, 2015) para dibujar utilizando JavaScript. Por ejemplo, el Código 1.1 muestra como se dibujaría una línea con esta Application Programming Interface (API). Los gráficos acaban siendo también imágenes rasterizadas, por lo que tenemos los mismos inconvenientes que con los *tiles*.

Código 1.1: Ejemplo de uso de la Canvas API para dibujar una línea

```
let c = document.getElementById("myCanvas");
let ctx = c.getContext("2d");
ctx.moveTo(0, 0);
ctx.lineTo(200, 100);
ctx.stroke();
```

Por último, SVG representa los gráficos en formato Extensible Markup Language (XML), haciendo

que cada elemento sea una forma, cuyo tamaño y posición se almacena de forma vectorial. De esta forma, se pueden escalar sin pérdida de calidad (al contrario que las imágenes rasterizadas). Si queremos aplicar las directrices de accesibilidad a mapas en línea, es prácticamente obligatorio utilizar un renderizador de SVG[1]. Esto se debe a que los atributos ARIA pueden ser añadidos a los elementos XML que conforman el gráfico, al contrario que con las otras tecnologías, que representan las imágenes de otra forma. El interés por SVG para la accesibilidad no es algo nuevo (Campin, McCurdy, Brunet & Siekierska, 2003), pero es ahora cuando se considera alcanzable, ya que antes los navegadores web no eran compatibles con esta tecnología.

El resto de este Trabajo Final de Grado (TFG) está distribuido de la siguiente forma. En la siguiente sección, se comentará una parte de las investigaciones anteriores. Después, se arroja algo de luz en varios conceptos relacionados. Las tecnologías, arquitectura, diseño e implementación final de la solución es descrita en la cuarta sección, así como la evaluación de la accesibilidad del proyecto. Finalmente, la quinta sección discute los el resultado y sus posibles mejoras. También ofrece un vistazo al panorama actual de la accesibilidad en las nuevas tecnologías, así como sus implicaciones en el desarrollo de software moderno.

[1]La versión actual es la 1.1 (World Wide Web Consortium, 2011), aunque la 2.0 ya está en desarrollo

2 Estado de la cuestión

El hecho de que no existe una implementación homogénea de los PA ha causado que se lleven a cabo distintos estudios. En este apartado, comentaremos algunas de las investigaciones existentes, además de explicar el desarrollo de nuestro propio estudio sobre lectores de pantalla.

2.1 Investigaciones previas

Una de las investigaciones condujo varias pruebas en MacOS X y Windows 7 utilizando Jaws, NVDA y VoiceOver, los lectores de pantalla más populares (Ferraz, 2017). Los resultados son muy heterogéneos, probando la falta de consistencia entre implementaciones. El estudio concluye diciendo que el atributo `aria-label` de ARIA (World Wide Web Consortium, 2018a) junto al elemento `<text>` son leídos por la mayoría de los navegadores. También hace comentarios interesantes sobre la Search Engine Optimization (SEO), señalando que los elementos `<desc>` de SVG son indexados con éxito por Duck Duck Go, Bing, Yahoo y Google. Esto significa que será beneficioso utilizar los atributos ARIA junto al elemento `<desc>` para tener compatibilidad con los PA mientras se optimiza para los motores de búsqueda.

Otra investigación fue realizada sobre diagramas de flujo en SVG (Watson, 2018). Los gráficos tratados en esta investigación no son mapas en línea, pero como también son elementos gráficos, los resultados pueden ser aplicados a los mapas web. El estudio pretende proporcionar diagramas de flujo con semántica y navegabilidad. Con ese fin, utiliza una combinación del rol `img` y del atributo `aria-hidden` para ocultar los elementos no relevantes a los PA. También utiliza los roles `list` y `listitem` para mantener relaciones con significado entre los elementos del diagrama.

Otra investigación (Calle-Jimenez, Eguez-Sarzosa & Luján-Mora, 2019) propone una arquitectura para mapas web accesibles. Esta arquitectura consiste en tres elementos. El primero es la base de datos, que almacena datos geográficos y del mapa. El segundo es el servidor web, que da acceso a los datos. Finalmente, el componente de lado de usuario muestra el mapa al usuario. Estos componentes fueron suficiente para implementar con éxito un mapa web accesible, probado con usuarios reales. El test concluyó que la aplicación era fácil de usar y probó que la accesibilidad se puede conseguir. Los

usuarios comentaron la confusión que provoca la forma en la que el lector de pantalla pronuncia las palabras. Este estudio también enfatiza en la importancia de los metadatos. Esta clase de información es esencial para hacer mapas accesibles, porque es lo que los PA necesitan para informar al usuario de lo que ocurre en el sitio web.

Otra investigación propone un algoritmo que genera instrucciones automáticamente para personas con problemas de visión. Las instrucciones tienen el objetivo de orientar a estas personas por el interior de un edificio, haciéndoles llegar desde un punto inicial a uno final. El algoritmo consta de tres pasos:

1. Se genera un grafo dirigido donde los nodos son *landmarks*, es decir, puntos en el edificio con receptores Near Field Communication (NFC)[1]. El peso de las aristas será la distancia física entre ellos, aumentándolo a infinito si el camino presenta algún problema o disminuyéndolo si incluye alguna preferencia del usuario (ascensores, escaleras...).

2. Utilizando el algoritmo de Dijkstra (Cormen, Leiserson, Rivest & Stein, 2001) se obtiene el camino más corto entre el punto de partida y el final. Al ajustar los pesos en los casos mencionados anteriormente, se tendrán en cuenta para obtener el camino.

3. Se generan las instrucciones que guiarán al usuario para que haga el recorrido, utilizando un vocabulario acordado con una comisión para invidentes de Massachusetts.

Hay más proyectos de navegación para personas con discapacidades, por ejemplo, NavCog (Ahmetovic et al., 2016). En este artículo, aunque se habla de distintas formas de localizar al usuario en interiores, como los tags RFID (por radiofrecuencia), las redes WiFi disponibles en el edificio o la comunicación con luz visible (VLC), finalmente utilizan *beacons* de Bluetooth Low Energy. Las ventajas que enumera son:

- No hay que hacer cambios estructurales en el edificio.
- Los *beacons* son más precisos que otras técnicas.
- La localización se calcula en el dispositivo y no se comparte, lo que fomenta la privacidad.

El sistema cuenta con un servidor de mapas, donde se cargan los mapas en sí. Después, se carga la información de los *beacons* y la de accesibilidad. El sistema utiliza también un grafo para calcular las rutas. Respecto al guiado, el artículo comenta que en espacios abiertos sin referencias (por ejemplo, paredes) el usuario puede desviarse.

[1]Encontramos un problema en que los receptores sean NFC en vez de Bluetooth, ya que los primeros requieren una proximidad muy elevada para que la conexión se realice. Pensamos que los receptores Bluetooth tienen un alcance más adecuado para este tipo de propósitos, además de ser baratos.

La información del servidor es utilizada junto a las señales BLE y el giroscopio para guiar al usuario. Esto se hace en la app de NavCog para iOS. Destaca lo siguiente:

- Se utilizan metros para dar las instrucciones, aunque experiencias pasadas sugieren que las personas con baja visión preferían las instrucciones en pasos.
- Se utiliza un sonido repetitivo cuyo ritmo aumenta conforme más se acerca el usuario al destino, como guía para saber la distancia a éste.

Otro proyecto interesante es el llamado GRACES (Gómez Sáez, 2017). GRACES es un proyecto desarrollado por la Universidad Politécnica de Madrid, que fue fundado por Indra y la Fundación Universia. Su propósito es proporcionar accesibilidad en el desarrollo de software, concretamente en los diagramas que describen el sistema a desarrollar. La solución que proponen se centra en utilizar los modelos subyacentes a esos diagramas para ofrecer una representación alternativa que pueda ser explorable por personas invidentes. Este proyecto tiene similitud con el nuestro: intentan hacer accesibles elementos gráficos. Sin embargo, la gran diferencia entre la información que subyace a los gráficos que tratamos en cada proyecto (la información geográfica es muy diferente a un diagrama UML) hace que en la práctica no podamos aplicar mucho de GRACES en nuestro proyecto. Los autores pretenden utilizar modelos subyacentes para dar una representación alternativa a los diagramas. En este sentido, los autores han investigado varias herramientas que dan su propia solución a la accesibilidad en diagramas UML. GRACES es un proyecto que, aunque trata el mismo problema que el nuestro (la accesibilidad de gráficos), el campo en el que se está aplicando es tan distinto que hace difícil trasladar algo de un proyecto a otro.

Estos estudios asumen que los desarrolladores del mapa en línea están produciendo su propio renderizador con sus propios datos. Sin embargo, una forma común de mostrar un mapa en un sitio web es hacer uso de una API (por ejemplo, Google Maps API) y obtener los datos, cuando sean necesarios, de un tercero.

Un estudio muestra que los mapas embebidos de Google Maps no son accesibles (Logan, 2018). Esto ha sido un problema durante mucho tiempo, pero Google no tiene intención de resolverlo, aparentemente (“Accessibility of Google Maps API, Google Issue Tracker”, 2017). El estudio expone las modificaciones que son necesarias realizar al visualizador de mapas de Google para que sea accesible. En concreto:

- La API de Google Maps no provee navegación con teclado. Hay que implementarla utilizando el atributo `tabindex="0"`. También se debe modificar la presentación de los marcadores cuando reciben el foco, utilizando CSS.

- Si el mapa tiene demasiada información y, por lo tanto, demasiadas paradas para el teclado, puede resultar incómodo (o incluso imposible) de navegar. Es recomendable agrupar los marcadores del mapa para que las paradas de teclado no excedan de las veinte.
- Hay que utilizar `aria-label` y `role="button"` para dar semántica a los marcadores y que los lectores de pantalla puedan leer correctamente lo que representan.
- Es necesario mostrar una representación textual alternativa que pueda ser navegable y filtrable a través de un cuadro de búsqueda.

Sobre Google Maps también existe otro artículo (Aoyama, 2019), en el que se trata la usabilidad de la aplicación de Google Maps en una serie de *Do* y *Don't*:

- Mantener una interfaz sin sobrecarga de información que permita cambiar a un modo con información minimizada. Las opciones que más utilizan los usuarios con problemas de visión son: búsqueda, cómo llegar a casa, llamar ayuda, modo *offline* y silenciar mapa.
- Utilizar nombres de elementos conocidos en vez de nombres de calles, así como evitar dar direcciones sólo con el tiempo o distancia hasta allí.
- Permitir a los usuarios ver todos los pasos y navegarlos.
- La forma más efectiva de comunicar dirección es leer la guía de navegación de la ruta entera y, después, notificar cada 150 — 450 metros.
- Poder utilizar los botones físicos para conocer la ubicación actual o para abrir Siri / Google Assitant.

2.2 Estudio de combinación de ARIA con navegadores y lectores

Llevamos a cabo nuestro propio estudio en el que investigamos qué combinaciones de atributos ARIA y navegadores funcionan mejor con los lectores de pantalla y SVG, con el objetivo de conocer la mejor estrategia de accesibilidad para nuestra aplicación. En él, consideramos los lectores de pantalla NVDA y Chromevox. El primero se probó en Firefox, Edge e Internet Explorer, mientras que el segundo fue probado en Chrome. En la Tabla 2.1 se detallan los navegadores y lectores utilizados junto a sus versiones. En la Tabla 2.2 se detallan las versiones de los lectores de pantalla utilizadas.

El estudio consistió en probar, en cada una de las combinaciones antes mencionadas (NVDA en Firefox, Edge e Internet Explorer; Chromevox en Chrome) diversos elementos de SVG con atributos ARIA. En concreto, se presta atención a elementos como `title` o `desc` (entre otros), que sirven

Navegador	Versión
Mozilla Firefox	61.0
Google Chrome	67.0
Microsoft Edge	42.17134
Microsoft Internet Explorer	11.112

Tabla 2.1: Detalle de los navegadores utilizados en el estudio

Lector	Versión
NVDA	2018.2.1
Chromevox	53

Tabla 2.2: Detalle de los lectores de pantalla utilizados en el estudio

para proporcionar títulos y descripciones de elementos. También se investigan atributos como `role`, `aria-label`, `aria-labelledby`, `aria-describedby` y `tabindex`. Una relación completa de las combinaciones se puede observar en la Tabla 2.3. El Código 2.1 fue utilizado en la realización de este estudio.

Código 2.1: Página HTML utilizada en el estudio de lectores y navegadores

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Documento HTML de prueba</title>
<meta name="description" value="Descripción del documento" />
<meta charset="utf8" />
</head>
<body>
<h1>Encabezado del documento</h1>

<!-- aria-describedby="figcaption" -->
<figure>
<!-- aria-labelledby="title" aria-describedby="desc" -->
<svg width="300" height="300" version="1.1" xmlns="http://www.w3.org/2000/svg" tabindex
    ↪ ="0">
<title id="title">Mapa en formato SVG</title>
<desc id="desc">Descripción del mapa en formato SVG</desc>

<g role="list" tabindex="0">
<title>Título de los elementos</title>
```

```
<desc>Descripción de los elementos</desc>

<a xlink:href="#" aria-label="Título del enlace 1" role="listitem">
<text x="10" y="20">Enlace 1 con aria-label </text>
</a>

<a xlink:href="#" title="Título del enlace 2" role="listitem" tabindex="0">
<text x="10" y="50">Enlace 2 con title</text>
</a>

<rect aria-label="Rectángulo" x="200" y="10" width="10" height="10" tabindex="0" />

<a tabindex="0">
<text x="10" y="80">No es enlace 1 con tabindex</text>
</a>

<a role="img">
<text x="10" y="110">No es enlace 2 con role="img"</text>
</a>

<a role="img" tabindex="0">
<text x="10" y="140">No es enlace 3 con tabindex y role="img"</text>
</a>

<a role="presentation" aria-hidden="true">
<text x="10" y="170">Enlace no accesible</text>
</a>

<a xlink:href="#" aria-labelledby="ltitle" aria-describedby="ldesc">
<title id="ltitle">Título de enlace 3 con elemento title</title>
<desc id="ldesc">Descripción del enlace 3</desc>
<text x="10" y="200">Enlace 3 con elemento title</text>
</a>
</g>
</svg>

<figcaption id="figcaption">
Descripción de la figura
```

```
</figcaption>
</figure>
</body>
</html>
```

En la Tabla 2.3 se pueden ver los comportamientos que se esperan de cada combinación, según las exigencias de la especificación ARIA. Solo hemos considerado Chromevox en Chrome, ya que con el resto de navegadores no es compatible.

Cuando se realizan las pruebas pertinentes, se obtienen los siguientes resultados, detallados en la Tabla 2.4. De ellos concluimos lo siguiente.

- Es interesante ver como lectores que son conocidos por funcionar muy bien en ciertos navegadores, por ejemplo NVDA y Firefox, empiezan a fallar cuando se trata de elementos contenidos en SVG. Esto es muy importante a la hora de desarrollar el mapa web ya que saber a qué carencias nos enfrentamos en cada navegador nos permite optimizar el mapa lo máximo posible para todos ellos.

- La concentración de fallos en los enlaces (lo único que permite dar interactividad a los gráficos SVG) nos indica que debemos llevar sumo cuidado y asegurar una buena accesibilidad, probándolos profundamente a lo largo del desarrollo.

- Podemos ver que el panorama general es complicado. Nos encontramos en una situación parecida al comienzo de la Web, cuando cada navegador interpretaba el código HTML a su manera: cada lector interpreta también lo que quiere y tienen problemas de compatibilidad entre navegadores. Es algo que seguramente se solucione con el tiempo y con un esfuerzo de estandarización, pero no está previsto que esto ocurra temprano.

- Habría que considerar la posibilidad de desaconsejar el uso del mapa en Internet Explorer, ya que ni siquiera ha permitido recorrer los elementos con el teclado, lo que supone una barrera de accesibilidad demasiado alta como para que sea tolerable.

- Se aprecia que la mejor forma de incluir un gráfico SVG en hacerlo de forma *inline*, ya que permite mejor navegabilidad con el teclado. El comportamiento del SVG embebido depende tanto del elemento que se use para ello (`object`, `embed`, `iframe`, `img`...) y del navegador que resulta imposible, en estos momentos, escoger un método que funcione en todas circunstancias.

El estudio concluyó con algunos atributos que tienen el mismo efecto en Chrome, Firefox y Edge.

Elemento	Comportamiento esperado
`<figure>` con `<figcaption>`	El lector lee el texto contenido en `<figcaption>` cuando se enfoca el mapa
`<figure>` con `<figcaption>` y `aria-describedby`	El lector lee el texto contenido en `<figcaption>` cuando se enfoca el mapa
`role="group"`	Permite que el lector de pantalla pueda recorrer los elementos del mapa
`<title>`	El lector de pantalla utiliza el texto contenido en `<title>` para enunciar el título del mapa
`<desc>`	El lector de pantalla utiliza el texto contenido en `<desc>` para enunciar la descripción del mapa
`<title>` con `aria-labelledby`	El lector de pantalla utiliza el texto contenido en `<title>` para enunciar el título del mapa, y esto ocurre una sola vez
`<desc>` con `aria-describedby`	El lector de pantalla utiliza el texto contenido en `<desc>` para enunciar la descripción del mapa y esto ocurre una sola vez
`<a>` con `aria-label`	El lector de pantalla reconoce un enlace a otro documento y utiliza la etiqueta `aria-label` para ampliar la información proporcionada por el contenido de la etiqueta
`<a>` con `title`	El lector de pantalla reconoce un enlace a otro documento y utiliza la etiqueta `title` para ampliar la información proporcionada por el contenido de la etiqueta
`<a>` que no es enlace con `tabindex="0"`	El lector de pantalla no debería enunciar como un enlace un elemento `<a>` que sólo está presente para dar interactividad al mapa
`<a>` que no es enlace con `role="img"`	El lector de pantalla no debería enunciar como un enlace un elemento `<a>` que sólo está presente para dar interactividad al mapa
`<a>` que no es enlace con `role="img"` y `tabindex="0"`	El lector de pantalla no debería enunciar como un enlace un elemento `<a>` que sólo está presente para dar interactividad al mapa
`role="list"`	El lector de pantalla considera los elementos contenidos en un elemento con este role como elementos de una lista, siempre y cuando estén etiquetados con `role="listitem"`
`role="presentation"` y `aria-hidden="true"`	El lector de pantalla ignora elementos con estos dos atributos
`<title>` y `<desc>` en un elemento gráfico (referidos con `aria-`)	El lector de pantalla utiliza el texto contenido en `<title>` para enunciar el título y la descripción del elemento gráfico
`<title>` y `<desc>` en una agrupación (referidos con `aria-`)	El lector de pantalla utiliza el texto contenido en `<title>` para enunciar el título y la descripción del elemento gráfico
Incluido desde un archivo externo	El lector se comporta igual que en las pruebas anteriores cuando el SVG se incluye desde un fichero externo

Tabla 2.3: Relación de cada combinación de elementos y atributos con el corportamiento esperado según la especificación ARIA

	Navegador				
	Firefox	**Chrome**		**Edge**	**Internet Explorer**
Elemento	**NVDA**	**NVDA**	**Chromevox**	**NVDA**	**NVDA**
`<figure>` con `figcaption`	F	F	F	F	F
`<figure>` con `figcaption` y `aria-describedby`	F	F	F	F	F
`role="group"`	T	T	T	T	F
`<title>`	T	T	T	T	T
`<desc>`	T	T	T	T	T
`<title>` con `aria-labelledby`	T	T	T	T	T
`<desc>` con `aria-describedby`	T	T	T	T	T
`<a>` con `aria-label`	T	T	T	F	F
`<a>` con `title`	F	F	F	F	F
`<a>` que no es enlace con `tabindex="0"`	F	F	F	F	F
`<a>` que no es enlace con `role="img"`	T	T	T	T	F
`<a>` que no es enlace con `role="img"` y `tabindex="0"`	F	F	T	T	F
`role="list"`	F	T	T	T	F
`role="presentation"` y `aria-hidden="true"`	T	T	T	T	F
`<title>` y `<desc>` en un elemento gráfico (referidos con `aria-`)	T	T	T	T	T
`<title>` y `<desc>` en una agrupación (referidos con `aria-`)	F	F	F	F	F
Incluido desde un archivo externo	F	F	F	T	F
Porcentaje de cumplimiento (%)	**64.29**	**71.43**	**78.57**	**57.14**	**35.71**

Tabla 2.4: Resultado de probar las diferentes combinaciones en los diferentes navegadores y lectores. La letra T indica que cumple con el resultado esperado, mientras la letra F indica que no lo hace

1. El rol `presentation` junto al atributo `aria-hidden` puede ocultar un elemento al lector de pantalla en los tres navegadores.

2. El rol `group` parece funcionar correctamente en los tres navegadores.

3. Los elementos `title` y `desc` exponen el título y la descripción de una forma correctamente al lector de pantalla en los tres navegadores.

4. El atributo `aria-label` funciona en Firefox y Chrome, pero no en Edge.

Concluimos que la mejor opción es escribir código que utilice el elemento `desc` y atributos ARIA, para obtener la máxima compatibilidad en los tres navegadores, mientras aseguramos una buena indexación en los motores de búsqueda. Esta conclusión se acerca a aquella que aparece en otros estudios sobre este tema (Ferraz, 2017; Fisher, 2019).

3 Conceptos relacionados

Algunos conceptos relacionados con este TFG serán definidos en esta sección. Conceptos que ya han aparecido en el TFG serán definidos también, para mayor claridad.

Como se ha dicho anteriormente, la accesibilidad consiste en qué productos, sistemas, servicios, entornos e instalaciones pueden ser utilizadas por personas con el mayor rango de capacidades (International Organization for Standardization, 2014). Aplicado a los sitios web, esto significa que la efectividad, utilidad, usabilidad y eficiencia debería ser la misma para usuarios con y sin discapacidad (Brajnik, 2009). Es decir, el objetivo de la accesibilidad web es conseguir que personas con discapacidad utilicen el sitio no de la misma forma, sino con los mismos resultados y funcionalidades, preservando la usabilidad.

WCAG son una serie de directrices que pretenden proveer una referencia estándar de lo que los desarrolladores y diseñadores deben hacer para conseguir que un sitio web sea accesible. La última versión es la WCAG 2.1 (World Wide Web Consortium, 2018b). WCAG proporciona cuatro principios: perceptible (la información y la interfaz debe se perceptible a los usuarios), operable (los componentes de la interfaz y la navegación debe ser operable), comprensible (la operación y la información deberían ser comprensibles) y, finalmente, robusto (el contenido debe poderse interpretar por un gran rango de agentes de usuario, incluyendo PA). Para cada uno de estos principios, se presentan directrices para proporcionar una forma de alcanzarlos en cualquier sitio web. Cada directriz explica, brevemente, una característica que contribuye a la accesibilidad. Entonces, se muestran diferentes criterios, con distintos niveles (A, AA y AAA, en orden decreciente de importancia). Estos criterios son utilizados después para comprobar la conformidad. Para ser conforme con un nivel, los anteriores también deben cumplirse. Esto significa, por ejemplo, que para alcanzar la conformidad con el nivel AA, deben cumplirse las directrices de nivel A y AA.

SVG (World Wide Web Consortium, 2011) es un formato, basado en XML, que representa gráficos vectoriales. SVG provee dos características interesantes. En primer lugar, como los gráficos son vectores, pueden ser escalados sin pérdida de calidad. En segundo lugar, como SVG está basado en XML, utiliza etiquetas para representar gráficos. Esto nos permite añadir metadatos a cada elemento, y esto es clave para la accesibilidad. Como se ha explicado ya, la especificación ARIA proporciona

formas de mejorar la accesibilidad de un sitio web utilizando atributos y roles. Estos también pueden ser utilizados en elementos de SVG, para construir gráficos accesibles con la combinación de ambas tecnologías.

Otro concepto interesante es el de comportamiento percibido[1]. Este concepto está relacionado con las acciones que un usuario percibe que serán posibles, basándose en la apariencia de un objeto (Connor, 2010). Para que un gráfico SVG soporte navegación por teclado, los elementos deben ser envueltos en enlaces. Esta es la única forma, hoy en día, de hacer accesible por teclado un gráfico SVG (Migliorisi, 2016). Sin embargo, esos enlaces no funcionan realmente como tal, así que el comportamiento percibido de esos elementos se ve afectado. Una solución para esta situación es modificar la semántica del enlace utilizando el atributo "role" (Migliorisi, 2016). El comportamiento percibido es una propiedad muy importante cuando se desarrolla una aplicación accesible, porque sean cuales sean los trucos que utilicemos para lograr la accesibilidad, la apariencia y la semántica deben seguir relacionados. La funcionalidad de un elemento, incluso si se modifica por accesibilidad, debe ser idéntica a la impresión que el usuario tiene de las posibles acciones cuando lo ven.

Es importante destacar que, para que los elementos dentro del SVG puedan recibir el foco, la raíz del gráfico debe contener un atributo `tabindex` con valor cero. Este atributo hace que un elemento pueda recibir el foco o no, dependiendo de su valor. Solo cero (recibe foco por programa o por teclado) o menos uno (recibe foco solo por programa) deberían ser utilizados. Otros valores de este atributo están desaconsejados porque alteran el orden de navegación de los elementos (Watson, 2014; WebAIM, 2016). Este atributo es fundamental para la accesibilidad, porque permite a los desarrolladores hacer que cualquier elemento pueda recibir el foco, así que los usuarios pueden acceder a toda la información tabulando a través de la página.

[1] Traducción del autor del término "perceived affordance"

4 Objetivos

Este TFG tiene como objetivo general conseguir un sistema que haga accesible un mapa geográfico en línea. Para lograrlo, se tienen una serie de objetivos particulares, que deben cumplirse para lograr el principal.

1. Conseguir una formación teórica suficiente como para plantear más adelante un sistema real que resuelva el problema de la accesibilidad en mapas en línea. Esta formación se logra con el seguimiento de cursos y la lectura de artículos y recursos relacionados.

2. Alcanzar conocimientos técnicos de las tecnologías necesarias para construir el sistema. En este caso, se deben adquirir conocimientos de SVG, ARIA y las directrices WCAG.

3. Investigar los retos que enfrentan las personas discapacitadas en su uso diario de las nuevas tecnologías y analizarlos, comprendiendo los tipos de discapacidad existentes.

4. Recopilar toda la investigación teórica para que sirva como documentación del TFG y como justificación para las decisiones que se tomen más tarde en el planteamiento del sistema.

5. Realizar un prototipo inicial que sirva para explorar los conocimientos teóricos obtenidos, y realizar pruebas con los diferentes lectores de pantalla y tecnologías de asistencia.

6. Asegurar que el prototipo obtenido es una base sólida para desarrollar a partir de él la aplicación final

7. Plantear y diseñar los aspectos del sistema de mapas en línea antes de su implementación, prestando atención a la estructura de la base de datos y a la arquitectura de la aplicación.

8. Desarrollar la aplicación de forma incremental, realizando pruebas periódicas de lo implementado.

9. Realizar pruebas finales de la aplicación para comprobar su buen funcionamiento, así como documentar todo el proceso de desarrollo en la memoria del TFG.

5 Metodología

El desarrollo de este TFG tiene una forma de trabajo individual del alumno, pero siempre con el apoyo y supervisión del tutor.

Con la ayuda del tutor, el alumno realiza un estudio teórico del estado de la cuestión, investigando tanto por su cuenta como los recursos que el tutor le facilita. La información recopilada se sube a un blog con tres objetivos: tener todo el conocimiento en un solo sitio, que el tutor pueda supervisar el avance del alumno y que sirva como material de apoyo para la redacción del TFG.

Una vez el alumno considera que tiene el conocimiento suficiente para comenzar el desarrollo, se diseñan y planean tanto la estructura como la arquitectura de la aplicación y de sus datos. Todo el avance que se hace en el diseño se sube al blog, para su supervisión por parte del tutor. Se celebran también reuniones periódicas donde el alumno muestra el trabajo hasta el momento y el tutor sugiere mejoras o el siguiente paso a seguir. La mayoría de la comunicación entre alumno y tutor se realiza en línea, como comentarios en el blog y por correo electrónico.

También se intenta sacar el máximo provecho al trabajo realizado. Para ello, hablamos con otros departamentos de la Universidad para compartir conocimientos y buscar diversas opiniones sobre el TFG, con el objetivo de mejorarlo todo lo posible. En este sentido, también se redacta un artículo para la revista "Enfoque UTE"[1] (indexada en el "Emerging Sources Citation Index"[2] de "Web of Science"), que sirve a su vez como práctica para la redacción de la memoria final. El alumno aprende así las bases de la creación de un artículo científico, así como trucos útiles para la misma, además de obtener cierta experiencia en el campo de la investigación científica. Finalmente, el tutor guía al alumno en la redacción de la memoria y le ayuda también a la preparación de la defensa, proporcionando recursos y materiales útiles para la misma (Luján-Mora, 2014).

En cuanto a la parte técnica de la metodología, el proyecto se desarrolla siguiendo un modelo incremental, cuya cronología es relajada al contar con solo una persona. Se proponen varios hitos relacionados con el desarrollo del proyecto que, en resumen, son:

1. Plantear el diseño de la base de datos (tanto geográfica como de accesibilidad) y la arquitectura

[1] http://ingenieria.ute.edu.ec/enfoqueute/index.php/revista/index
[2] http://mjl.clarivate.com/cgi-bin/jrnlst/jlresults.cgi?PC=MASTER&ISSN=*1390-9363

a construir.

2. Construir un prototipo que sirva como base del proyecto completo, y como prueba de que es posible alcanzar los objetivos con el diseño pensado.

3. Presentar el prototipo al tutor y consensuar mejoras y correcciones.

4. Una vez obtenida una base sólida, se implementan características en forma de ampliaciones de dicho prototipo, probando periódicamente (normalmente cada vez que se introduce un cambio o nueva característica) el correcto funcionamiento de la aplicación. Esta fase se repite en forma de iteración hasta que la implementación está terminada.

5. Cuando la implementación está en un punto avanzado (lo que se deja a consideración del alumno), se despliega en un servidor de acceso público. Se eligen los AWS porque su capa gratuita ofrece las características necesarias para desplegar esta aplicación correctamente. Esto permite que el tutor pueda revisar desde casa el avance del alumno.

6. Una vez la implementación se da por terminada, se prueba la aplicación globalmente y se procede a la detección de fallos (y a su corrección) y a la mejora de las características o del rendimiento. A la vez, se confecciona la memoria del proyecto.

Cabe destacar que el diseño o la arquitectura no son fijas y pueden cambiar en cualquier momento dependiendo de los problemas que vayan surgiendo durante el desarrollo. Por ejemplo, el motor de visualización del mapa cambia a mitad de la fase de implementación porque se descubre una alternativa mejor a un visualizador propio: el *plug-in* Leaflet. La forma en la que se procesan los datos y se intercambian entre cliente y servidor cambian totalmente a raíz de esto, pero la organización del trabajo incremental y por iteraciones permite no sólo avanzar, sino realizar este tipo de modificaciones sin mayor problema.

Las distintas versiones que se van generando de la aplicación se gestionan con la utilidad Git y se almacenan, como copia de seguridad, en un repositorio privado de GitHub. De la memoria también se guarda una copia de seguridad en un repositorio distinto, también privado.

Por último, la cronología de trabajo para el desarrollo del proyecto es la siguiente. Se ha expresado en tiempo aproximado, porque no todas las semanas se ha dedicado el mismo esfuerzo.

Hitos 1, 2 25 días aprox.

Hitos 3, 4 7 meses aprox.

Hito 5 1 mes aprox.

Hito 6 1 mes aprox.

6 Desarrollo

Para resolver los problemas de accesibilidad que un mapa web normalmente tiene, hemos desarrollado una aplicación web que consisten en un componente *back-end* y en otro *front-end*. Esta aplicación web ofrece dos modos: el de exploración, que permite a los usuarios tener una visión general de la zona, y el de navegación, que calcula rutas entre dos puntos utilizando el algoritmo de Dijkstra (Cormen et al., 2001) y las muestra en el mapa.

6.1 Base de datos

Los datos geográficos están almacenados en una base de datos PostgreSQL[1], utilizando el *plug-in* PostGIS[2]. Esta elección se fundamenta en dos motivos. En primer lugar, en el proyecto hemos priorizado el uso de software libre siempre que ello sea posible. Esto se debe a que este tipo de software tiene características que son ventajosas para el proyecto. El tipo de licencia bajo el que se distribuye el software libre (Singh, Bansal & Jha, 2015) permite que los resultados de este TFG sean replicados por cualquier individuo o equipo sin incurrir en problemática de licencias o permisos de uso. Además, esto también permite desplegar el proyecto (como se explica en la Sección 6.4) sin problemas legales. Otra ventaja es el apartado económico, y es que el software libre es gratuito (Singh et al., 2015). Esto permite reducir los costes del proyecto enormemente, y definitivamente hace asumible su desarrollo al estudiante.

Dentro del software libre, era necesario escoger un motor para la base de datos. Hay diversos motores disponibles en el mercado hoy en día, como pueden ser MySQL, PostgreSQL, MariaDB, etc. Puede notarse que todos los motores citados siguen el modelo relacional. Creemos que, según la naturaleza de los datos que manejamos, que mantienen fuertes relaciones entre ellos, este modelo es el más adecuado. Por tanto, motores NoSQL como puede ser MongoDB quedan descartados.

Considerando las posibilidades respecto a la gestión de datos espaciales, contamos con el *plug-in* PostGIS en PostgreSQL y, por otro lado, con MySQL Spatial. Parece que PostGIS tiene una mejor implementación de los estándares (Piórkowski, 2011). Además, su robustez (al estar basado en

[1] https://www.postgresql.org/
[2] https://postgis.net

PostgreSQL), sus años de existencia y la gran comunidad detrás de él lo hicieron un candidato idóneo para nuestro proyecto.

El *plug-in* PostGIS nos permiten realizar consultas espaciales rápidas y avanzadas contra la base de datos. La información geográfica está organizada en capas, cada una correspondiente a una categoría, que son:

Edificio Los datos que representan a los edificios pertenecen a esta capa.

Obstáculo Esta capa almacena zonas difíciles de transitar.

Asistencia Esta capa almacena zonas que tienen ayudas para personas discapacitadas (por ejemplo, pavimento táctil).

Interés Lugares y puntos que remarcar en las rutas generadas.

Camino Un conjunto de caminos que pueden ser transitados a pie.

Entrada Lugares por los que un edificio puede ser accedido. También son consideradas salidas.

Los datos que se almacenan provienen de dos orígenes. Por un lado, tenemos aquellos generados manualmente durante el desarrollo del proyecto. Los obstáculos, asistencias, caminos y entradas son de este tipo. Por otro lado, se integran datos del SIGUA, de la Universidad de Alicante. Concretamente, se obtiene información geográfica de los edificios que conforman la Universidad, derivando de ahí también la información de la capa de interés.

Tanto los datos importados de SIGUA como los generados manualmente son gestionados mediante el software QGIS[3]. Las categorías de la base de datos corresponden cada una a una capa dentro de dicho software. Cada capa almacena información relacionada con su categoría. Manualmente, se utilizan las herramientas presentes en el programa para gestionar puntos, líneas o polígonos, según el tipo de información. El programa también provee utilidades de importación de datos que utilizamos para obtener, desde el servicio WFS que proporciona SIGUA para sus mapas, todos los datos geográficos de los edificios de la Universidad.

A la hora de cargar todos estos recursos en la base de datos, utilizamos la utilidad "shp2pgsql-gui". Nos permite, desde una interfaz gráfica, la creación y actualización de tablas que contengan cada una los datos contenidos en una capa. Requiere que previamente se haya generado un archivo *shapefile* para cada capa, pero esto puede hacerse fácilmente con QGIS. Así, la base de datos acaba conteniendo una tabla para cada capa, que a su vez corresponde a una categoría de información. Podemos observar una representación de cómo se relacionan y organizan en la Figura 6.1.

[3]https://www.qgis.org

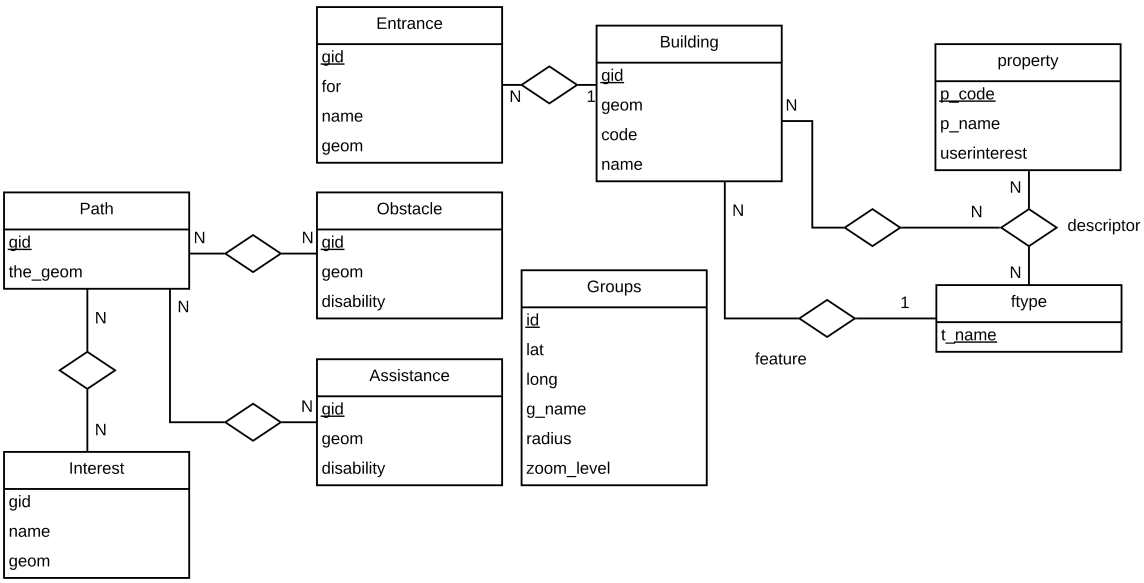


Figura 6.1: Diagrama entidad-relación que muestra la base de datos de la aplicación

Algo a tener en cuenta en la información geográfica es la proyección en la que está representada. Una mala gestión de la proyección puede alterar la geografía almacenada y arrojar resultados erróneos. En nuestro caso, hemos decidido conservar la proyección utilizada en SIGUA en el resto de capas, para mayor compatibilidad de unas con otras y para evitar conversiones de unidad (ya que algunas proyecciones usan metros y otras, grados). Por lo tanto, todas las capas se proyectan en EPSG:25830. Cabe destacar que esta proyección en concreto sólo es válida en una zona de Europa (MapTiler Team, 2017). Si quisieramos mostrar otros lugares, habría que recurrir a otra, como por ejemplo la EPSG:4326 (MapTiler Team, 2007), que es global y utilizada por los sistemas GPS.

La combinación de estas seis capas resulta en información suficiente como para construir un sistema que provea información útil para usuarios discapacitados. Esto se logra implementando una aplicación web con la que los usuarios puedan interaccionar. Con este propósito, utilizamos un servidor NodeJS[4] y otro Apache[5].

6.2 Arquitectura

Un servidor NodeJS maneja la parte del *back-end* y se encarga de interaccionar con la base de datos, para acceder a sus datos y proveerlos al otro servidor. Por seguridad, los clientes se comunican con el servidor Apache bajo el protocolo Hypertext Transfer Protocol Secure (HTTPS) y el servidor NodeJS solo es accesible desde el servidor Apache. Los resultados obtenidos en las consultas a la base

[4]https://nodejs.org
[5]https://httpd.apache.org/

de datos se devuelven en formato GeoJSON (Internet Engineering Task Force, 2016). Este formato está definido por la Internet Engineering Task Force (IETF) y se utiliza para representar y codificar datos geográficos utilizando JavaScript Object Notation (JSON). NodeJS ha sido utilizado como el servidor *back-end* porque ofrece una forma fácil y confiable de manejar peticiones, junto a una amplia comunidad y librerías. Otras tecnologías podrían haber sido utilizadas (como PHP o ASP), pero NodeJS es más ligero y, por lo tanto, tiene un mejor rendimiento bajo servidores modestos.

Otro servidor Apache maneja el componente *front-end* y tiene la responsabilidad de aceptar las peticiones y responder a ellas, así como de almacenar y servir archivos estáticos (*scripts*, hojas de estilo, iconos, archivos HTML, etc.). Esta es la parte más importante de la solución: para que la información almacenada sea útil, debe ser presentada de forma accesible.

La comunicación entre los dos servidores se realiza mediante peticiones HTTP no encriptadas. Esto no supone un riesgo de seguridad porque la versión de desarrollo se prueba en local y la versión de producción tiene correctamente controlados mediante Access Control List (ACL) qué conexiones pueden realizarse, como se explicará más adelante.

Para cada operación que se desee realizar con los datos de la base de datos, se crea un URL plantilla en el servidor NodeJS a la que poder hacer una petición HTTP desde el otro servidor. Se detallan en la Tabla 6.1 los Uniform Resource Locator (URL) disponibles.

Estas peticiones devuelven todas un resultado en formato JSON. Algunas de ellas contienen a su vez datos en GeoJSON, siempre que sea necesario devolver información geográfica.

En la parte de *front-end*, toda la programación en JavaScript es realizada en primer lugar en TypeScript. Esta decisión se tomó porque TypeScript permite una gestión de los tipos más avanzada, lo que nos permite tener una mayor seguridad a la hora de realizar cálculos complejos (sobre todo en el modo navegación), además de permitir declarar interfaces que también nos serán útiles en partes como el orden de tabulación de edificios, como se explicará más adelante en este TFG. El código escrito en TypeScript debe compilarse a JavaScript antes de poder ser ejecutado en el cliente, lo que se consigue con la herramienta `tsc`. Para facilitar las tareas de desarrollo, se construyó un archivo *makefile* (Código 6.1) que aglutina todas las tareas relacionadas con la gestión del código y su compilación.

También se puede observar como el archivo contiene órdenes para el empaquetamiento del código. Esta forma de entregar el código al cliente está pensada para mejorar el rendimiento de la aplicación, concretamente el tiempo de carga. Además, combina los módulos JavaScript de forma que no es necesario utilizar el atributo `type="module"`, lo que aumenta la compatibilidad con distintos navegadores. Como empaquetar el código es un proceso más lento que la compilación, su uso está previsto solamente para la puesta en producción. El código se empaqueta utilizando la utilidad Webpack, que unifica todo el código JavaScript en un único archivo que es transmitido al cliente durante la carga.

URL	Resultado
/map/data/geojson/:r	Datos geográficos (en GeoJSON) y de accesibilidad de cada edificio. Para cada edificio, adjunta también los más cercanos dentro del radio :r
/map/data/b/:id	Información geográfica y de accesibilidad sobre el edificio con identificador :id
/map/data/s/name/:name	Edificios cuyo nombre contenga :name
/map/data/p/:id1,:id2,:disability	Ruta desde el edificio con identificador :id1 hasta el edificio con identificador :id2 teniendo en cuenta la discapacidad :disability
/map/data/pi/:id1,:id2,:disability	Ídem al anterior, pero adjuntando puntos de interés cerca de cada punto de giro
/map/data/nn4f/:bid,:radius	Edificios que se encuentran dentro de un radio :radius alrededor de otro edificio con identificador :bid
/map/data/nn4p/:lat,:long,:radius	Edificios que se encuentran dentro de un radio :radius alrededor de las coordenadas (:lat, :long)
/map/data/rsvg	Código SVG que dibuja las rutas preguardadas en la base de datos
/map/data/tab/we	Lista con los identificadores de los edificios, ordenados de oeste a este
/map/data/:radius	Datos geográficos (directamente en SVG) y de accesibilidad de cada edificio
/map	Interfaz gráfica del modo de exploración
/route	Interfaz gráfica del modo de navegación
/settings	Interfaz gráfica de la pantalla de ajustes

Tabla 6.1: Detalle de los URL disponibles en el servidor back-end para interactuar con él y la base de datos

La convergencia del código en un solo archivo (y por lo tanto, una sola petición) junto a las técnicas de caché de los navegadores modernos, hace que el rendimiento mejore con esta técnica.

Una parte de las instrucciones del *makefile* están destinadas a la fase de desarrollo, y consisten en compilar el código TypeScript en JavaScript y hacer que esté disponible en archivos sueltos. Esto permite un desarrollo más ágil.

Código 6.1: Makefile que gestiona el código TypeScript y su compilación

```
# JAVASCRIPT BUILD
# Phases:
#  compile
#  Calls TypeScript compiler in order to compile the .ts files
#  into .js files
#
#  independent
#  Files in the ts/__independent folder are not meant to be bundled,
#  but to work independently as scripts loaded by the <script> tag.
#  They are copied direclty to the public/js folder for that purpose.
#
#  pack
#  The rest of the files are bundled in the way described in
#  webpack.config.js
#  Bundled files end up in the public/js/bundle folder
#

BLUE=\033[1;34m
CC=\033[0m # Clear Color

all: compile independent pack
debug: compile independent copy-debug

compile: folder-prepare
@echo "${BLUE}Compiling TypeScript files into compiled-js/ ${CC}"
tsc
@echo "\n"

independent:
@echo "${BLUE}Copying independent files to public/js ${CC}"
```

```
cp -r compiled-js/__independent/* public/js/
@echo "\n"

pack:
@echo "${BLUE}Creating bundles (defined in webpack.config.js)${CC}"
npx webpack --config webpack.config.js
@echo "\n"

copy-debug: folder-prepare
@echo "${BLUE}Copying compiled files to public/js for DEBUG ${CC}"
cp -r compiled-js/* public/js/
@echo "\n"

folder-prepare:
@echo "${BLUE}Preparing folder structure${CC}"
mkdir -p compiled-js/
mkdir -p public/js
@echo "\n"

clean:
rm -rf /public/js
rm -rf compiled-js
```

6.3 Interfaz y componentes

En este apartado se va a describir el diseño de la aplicación, las vistas que lo componen y la distribución de las mismas. La interfaz gráfica se ha realizado mediante el *framework* Bootstrap[6]. Entre otras opciones como Skeleton o Materialize, se escogío Boostrap por dos motivos. En primer lugar, permite crear de forma sencilla interfaces que se adapten al tamaño de pantalla en la que se muestran (*responsive*). En segundo lugar, la experiencia previa del alumno con el *framework* facilitaba el trabajo con el mismo.

La aplicación consta de tres vistas: el modo de exploración, el modo de navegación y los ajustes. El modo de exploración funciona también como página principal, es decir, es la vista que aparece si accedemos al URL de la aplicación.

Se ha tenido como propósito crear una serie de interfaces funcionales y modernas que además se

[6]https://getbootstrap.com/

Figura 6.2: Comparación de la interfaz en una pantalla de ordenador (más ancha) y una pantalla de teléfono móvil (más estrecha)

adapten a la pantalla en la que se muestran. Una comparación de la misma vista (la del modo de exploración) entre una pantalla de ordenador y otra de móvil puede verse en la Figura 6.2. Como se observa, a la izquierda aparecen todos los menús y componentes desplegados aprovechando el ancho de la pantalla. A la izquierda, los componentes se han replegado para poder mostrar el mapa y se pueden mostrar mediante el botón que aparece en la parte superior izquierda.

6.3.1 Modo de exploración

El primer modo que se desarrolla es el de exploración. La parte esencial de este modo es el mapa basado en un visualizador SVG. El modo de navegación también utilizará este visualizador, pero con funcionalidades distintas (lo que se explica en su correspondiente apartado 6.3.2).

6.3.1.1 Visualizador de mapas

Las primeras versiones de la aplicación contienen un visualizador propio, que obtiene SVG ya generado desde la base de datos PostGIS y lo inserta en un elemento SVG utilizando la librería SVG.js[7]. Las acciones como mover el mapa o acercarlo y alejarlo se implementan a mano. Por ejemplo, para manejar los niveles de *zoom*, llegamos a nuestra propia ecuación, cuyos valores tomamos de forma empírica observando cómo Google Maps traducía los metros a píxeles. La representación con menor zoom utiliza 0.000246153846 píxeles por metro y la de mayor, 8.578143361 píxeles por metro. Esto significa que para 20 niveles, cada uno cambiará en 0.4514682741 el valor. Esto nos deja con la siguiente fórmula:

[7]https://svgjs.com/

$$2.46153846 \cdot (10^{-4}) + (0.4514682741 \cdot (z-1))$$

Donde z es el nivel de *zoom* (de 1 a 20). El ancho del elemento SVG se divide entre el número obtenido por la fórmula y se obtiene finalmente el nuevo valor de la propiedad `viewbox`. Esta propiedad nos permite definir qué parte del SVG original vamos a mostrar y a qué escala. La propiedad acepta cuatro valores: las coordenadas X e Y del punto de origen, el ancho de la caja y el alto de la misma. Por ejemplo, los valores `50 50 100 100` mostrarán el contenido del SVG original situado en el rectángulo con vértice superior izquierdo (50, 50) y vértice inferior derecho (150, 150). Esto nos permite, modificando tanto el punto de referencia como el ancho y el alto, cambiar la escala del SVG y la parte del mismo que se muestra.

Durante un tiempo, este era el visor que utilizamos para el proyecto. De funcionamiento algo tosco, el visor solo es compatible con el SVG devuelto por el servidor. Esta situación cambia tras la reunión con José Manuel Mira Martínez, de SIGUA. Mira nos introduce al visor Leaflet, explicándonos su funcionamiento y enfatizando en su popularidad actual. Tras la reunión, valoramos la adopción de Leaflet en nuestro proyecto y concluimos que, aunque supone un parón en la línea de desarrollo prevista, los beneficios que aporta son suficientes como para asumirlo. Se demuestra tiempo después que fue una buena decisión, ya que las funcionalidades de Leaflet, con su API extensible y su comunidad detrás, facilitan y agilizan ciertas tareas.

Cuando hicimos el cambio a Leaflet, nos despreocupamos de los aspectos de control del mapa (puesto que el *plug-in* ya los maneja por sí mismo) y pudimos enfocarnos más en las funcionalidades del modo de exploración. Nuestro proyecto, como se ha visto en la arquitectura (Apartado 6.2) consta de un componente *front-end* encargado de realizar las peticiones pertinentes al *back-end* (utilizando los URL expuestos en el Apartado 6.1) y mostrar los resultados de forma accesible. La relación entre estos dos componentes se modifica al cambiar al visor, puesto que tras investigar Leaflet, descubrimos que trabaja obteniendo datos en GeoJSON. Por lo tanto, cambiamos el servidor *back-end* para que devuelva los datos geográficos en formato GeoJSON (PostGIS da soporte para esto). Como se ha plasmado en este documento, para que un documento gráfico sea accesible, debe representarse utilizando SVG, que es justo lo que utiliza Leaflet para visualizar los mapas.

Aprovechando la opción que da la API de Leaflet de extender la funcionalidad de sus componentes, creamos una clase propia que sustituya a la que, por defecto, crea el SVG a visualizar. En concreto, nuestro visualizador encierra cada edificio que se dibuja en un enlace, algo que es indispensable para su accesibilidad. Además, también inyecta en el elemento SVG información de accesibilidad. La forma en la que la representación se hace accesible se describe en mayor profundidad en el Apartado 6.5.

Este visor de mapas es el que se utiliza tanto en el modo de exploración como en el modo de navegación, pero con fines distintos. Por eso, en cada modo tiene un nivel de interacción distinto. Mientras que en el primero el usuario puede moverse entre los edificios con el teclado o seleccionar los edificios para obtener más información, en el segundo modo el visor solo sirve para visualizar la ruta simulada, así que estas funciones no están disponibles.

A continuación, se explicará el desarrollo y la implementación de las funcionalidades del modo de exploración.

6.3.1.2 Cerca de ti

La sección "Cerca de ti" de la aplicación es muy relevante, ya que supone la alternativa textual al mapa, que debe estar presente para lograr la máxima accesibilidad. Cuando una persona sin discapacidad observa un mapa, utiliza la información visual para encontrar su posición y observar lo que se encuentra a su alrededor. Esta sección es la versión textual a este proceso, donde utilizando los sensores de localización del teléfono, podemos calcular qué edificios tiene cercanos el usuario.

En primer lugar, se obtiene la localización del usuario utilizando la API destinada a ese fin, y que en JavaScript se utiliza desde el objeto `navigator.geolocation`. Las coordenadas devueltas son expresadas en el sistema de referencia EPSG:4326 (MapTiler Team, 2007). Para utilizarlas en nuestro servidor, que por compatibilidad con los datos de SIGUA utiliza el EPSG:25830 (MapTiler Team, 2017), debemos hacer una conversión. Esto se logra fácilmente con la librería proj4[8].

En segundo lugar, una vez tenemos las coordenadas en el sistema correcto, las pasamos al servidor para que nos devuelva una lista de lugares cercanos. En este caso hemos decidido no hacer los cálculos en el cliente porque PostGIS proporciona operaciones espaciales muy rápidas que son idóneas para este tipo de peticiones. Como se puede ver en la Tabla 6.1, la petición debe enviarse a `/map/data/nn4p/x,y,r`, donde x e y son las coordenadas del usuario y r es el radio en metros alrededor del cual buscar edificios cercanos.

La respuesta del servidor, en JSON, proporciona una lista de lugares cercanos, con información como el nombre, la distancia del usuario al mismo, las coordenadas donde se encuentra y el radio de búsqueda que se ha utilizado. Esta respuesta es procesada por el cliente para crear una lista de lugares cercanos a él, que se muestra al lado izquierdo de la página. Un ejemplo de una respuesta puede verse en el Código 6.2, donde se muestra un solo resultado, aunque normalmente se devolverán varios, ordenados por distancia.

[8] https://proj4.org/

Código 6.2: Ejemplo de respuesta JSON a una petición sobre lugares cercanos al usuario

```
[{
    "iid":50,
    "iname":"E.U. Relaciones Laborales de Elda (adscrita)",
    "icenterx":692364.150170141,
    "icentery":-4260606.50524218,
    "radius":1200,
    "distance":1147.48851974744
}]
```

6.3.1.3 Elementos a la vista

Cuando una persona mira un mapa, puede fijarse en los lugares que tiene a su alrededor (como comentábamos en el Apartado 6.3.1.2) o puede también querer un vistazo general de una zona en concreto. Este apartado de la página, "Elementos a la vista", busca proporcionar a personas discapacitadas una lista de lugares que se muestran en el mapa en ese momento justo.

Con Leaflet es muy sencillo controlar qué ocurre en el mapa, dado que proporciona una serie de eventos a los que conectar funciones. En concreto, conectamos a los eventos `zoomend moveend` una función que recorre los edificios y averigua si están dentro de la visualización del mapa.

Averiguar esto es también sencillo con las operaciones que permite Leaflet. En una sola línea podemos lograrlo, como vemos en el Código 6.3.

Código 6.3: Devuelve verdadero si un edificio está dentro de la visualización del mapa en ese momento

```
isInview(e) {
    return this.map.getBounds().contains((e.getLatLng ? e.getLatLng() : e.getCenter()))
        ↪ ;
}
```

Aplicar esta función a cada edificio resulta en una lista de lugares que están a la vista, que posteriormente se muestra al usuario en la barra lateral de la aplicación.

6.3.1.4 Botones de acción

La aplicación web cuenta con tres botones de acción, relacionados con la ubicación, la orientación del usuario y el control por voz. A continuación se explica la funcionalidad de cada uno.

6.3.1.4.1 Botón de ubicación Hacer clic en este botón resulta en centrar el mapa en la ubicación del usuario. Para ello, se hace uso de la API destinada a ese fin y que en JavaScript se utiliza desde el objeto `navigator.geolocation`. Las coordenadas se devuelven en un sistema de coordenadas que es compatible directamente con Leaflet. Sin embargo, cuando a mitad de proyecto se hizo el cambio de visor, se tomaron decisiones para mantener la compatibilidad del código.

La función encargada de mover el mapa a un punto en concreto (`SVGMap.instance.moveTo`) toma como parámetro las coordenadas del punto al que moverse. Antes de integrar Leaflet en el proyecto, este movimiento se calculaba con código propio utilizando el sistema EPSG:25830 (MapTiler Team, 2017). Leaflet utiliza EPSG:4326 (MapTiler Team, 2007) para las coordenadas del mapa, por lo que se decidió que la función siguiera tomando las coordenadas en el primer sistema y las transformara al segundo utilizando la librería proj4.

Cuando la API de localización nos devuelve las coordenadas, las transformamos al sistema adecuado y se las pasamos a la función que acabamos de describir. En el Código 6.4 se aprecia este proceso.

Código 6.4: Obtiene la localización actual y mueve el mapa a ella

```
1 locationService.getCurrentPosition(function(lat, long) {
2     let [x, y] = (<any>proj4('EPSG:4326', 'EPSG:25830', [long, lat]));
3     SVGMap.instance.moveTo(x, -y);
4 });
```

6.3.1.4.2 Botón de orientación Hacer clic en este botón hace que la información sobre orientación disponible (y que se muestra junto a él) reciba el foco, haciendo que el lector de pantalla la enuncie. Esta información sobre orientación se obtiene a partir de los sensores del dispositivo. En concreto, utilizamos la API que accede a la brújula para saber la orientación respecto al norte.

La orientación del usuario se calcula dibujando tres líneas imaginarias en el mapa, como puede verse en la Figura 6.3. Una línea apunta directamente hacia donde el usuario mira; las otras apuntan cada una al mismo punto pero con $\pm 10^{\circ}$ de diferencia, así que se usa un ángulo total de 20° para calcular la orientación, para compensar la imprecisión de las brújulas de los teléfonos móviles.

El sistema que hemos diseñado utiliza un sistema de votación para averiguar hacia dónde está mirando el usuario. Existen tres votos posibles y cada una de las líneas dibujadas cuenta con un voto. Cuando una de las líneas intersecta con un edificio, un voto para ese edificio es emitido. Las líneas tienen un alcance de 300 metros. Si en esa distancia no intersectan con ningún edificio, se emite un voto en blanco. Según la cantidad de votos en blanco, podemos tener diferentes casuísticas.

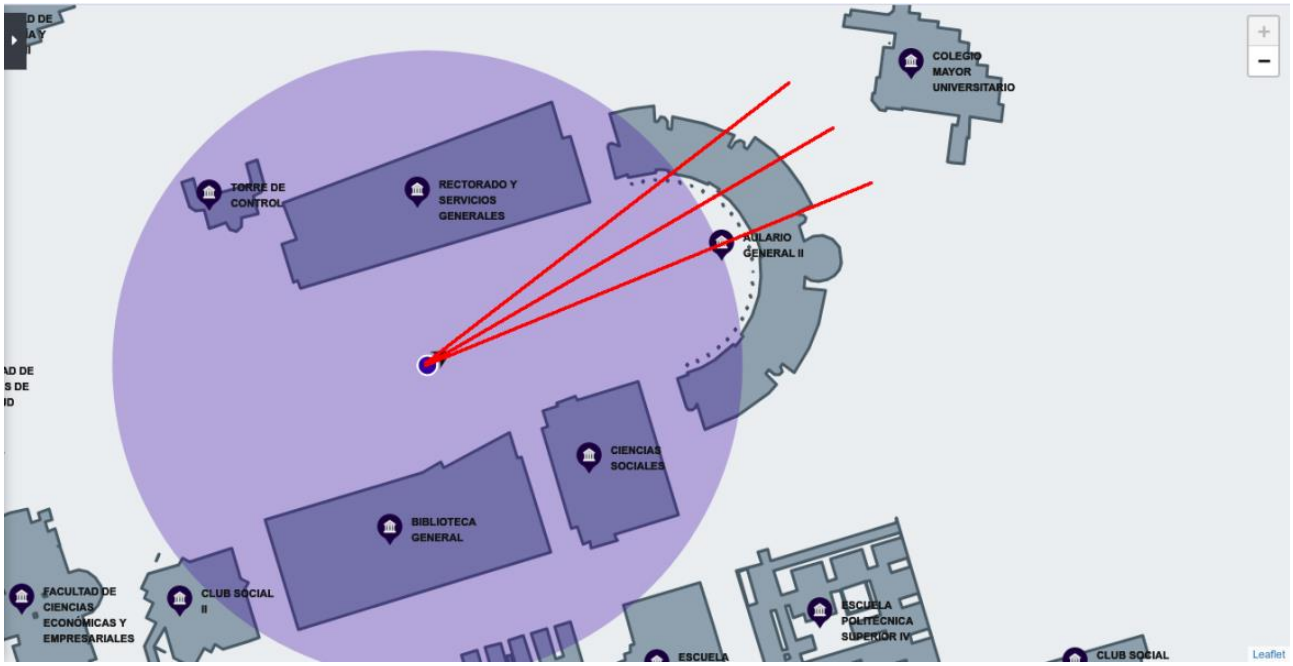


Figura 6.3: Técnica de aproximación de la orientación. Las tres líneas tienen 10° de diferencia entre ellas y votarían por el mismo edificio en este ejemplo.

- Tres votos en blanco significa que el usuario no está mirando ningún edificio cercano, y por lo tanto no se muestra nada en la aplicación.

- Dos votos en blanco dejan la responsabilidad de averiguar la orientación a una sola línea. Encontramos esta situación improbable, ya que entre cada una de ellas hay una distancia pequeña (recordemos, de 10°), así que para que esto sucediera estaríamos hablando de edificios muy pequeños.

- Un voto en blanco deja dos votos válidos enfrentados. Esta situación supone la posibilidad de que cada voto se emita a un lugar distinto, y no haya consenso. En este caso, se escoge el edificio que antes haya recibido el voto. Las líneas votan en orden creciente. Es decir, si la línea central mira hacia los 80°, el voto que cuenta en primer lugar es el de la línea que mira hacia los 70°, después la de 80° y, por último, la de 90°.

- Ningún voto en blanco supone que cada línea ha intersectado con un edificio. Si hay mayoría, se escoge ese edificio como orientación del usuario. Si no lo hubiera, es decir, cada línea apunta a un edificio distinto, seguimos la misma estrategia que en el punto anterior (escogemos el edificio cuyo voto se haya emitido antes).

En la Figura 6.3 podíamos observar como las tres líneas intersectan con el Aulario General 2, por lo que las tres votan por este edificio y acaba seleccionándose como la orientación del usuario. En la

Figura 6.4: Aproximación de la orientación, donde dos líneas intersectan con el mismo edificio, mientras que la tercera no

Figura 6.4 observamos una situación distinta, en la que una de las líneas intersecta con otro edificio. En este caso, hay dos votos para Aulario General 2 y uno para Rectorado. Se seleccionaría Aulario General 2 como orientación del usuario. Por último, en la Figura 6.5 vemos que el giro ha acabado, y las tres líneas vuelven a intersectar con un mismo edificio, en este caso Rectorado. Se escoge este último como orientación del usuario, y así se refleja en la aplicación.

Hacer estos cálculos es algo que requiere de cierta potencia computacional. Como los sensores de la brújula arrojan actualizaciones con un tiempo minúsculo de diferencia entre ellas, descartamos algunas de ellas para que la aplicación no sobrecargue el dispositivo. En concreto, sólo tomamos una de cada cuatro muestras de orientación de la brújula, el resto se descartan.

6.3.1.4.3 Botón de control por voz Este botón activa el modo de control por voz, que dependiendo de la pantalla en la que nos encontremos, tiene un propósito u otro. El control por voz utiliza la la Web Speech API (World Wide Web Consortium, 2019) tanto para reconocer lo que el usuario dice como para enunciar frases. Cuando se hace clic en el botón, la clase `SVGVoiceControls` entra en acción. Esta clase tiene como responsabilidad mantener la escucha de comandos abierta, llamándose a sí misma

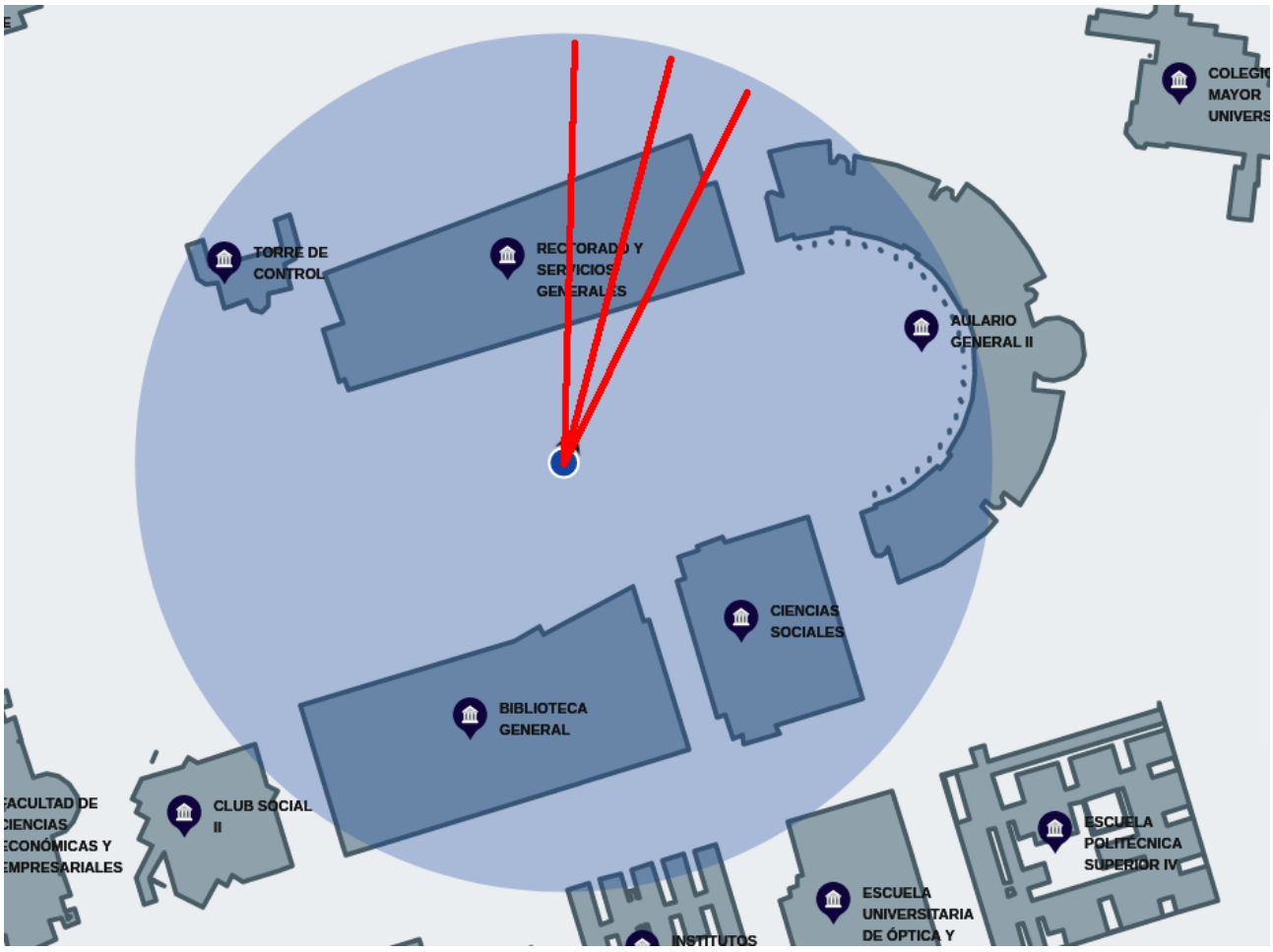


Figura 6.5: Aproximación de la orientación, cuando el giro ha completado y el usuario ahora está mirando hacia otro edificio. Así sería notificado por la aplicación

cuando sea necesario (en caso de que, por ejemplo, la API deje de escuchar por cualquier motivo). Solo dejará de escuchar cuando el usuario vuelva a hacer clic en el botón de control por voz.

La clase tiene también como responsabilidad gestionar las frases que se deben pronunciar en voz alta, para comunicarse con el usuario. Para ello, mantiene una cola de lo que hemos llamado `SpeechOrder`. Cada orden contiene el texto a pronunciar, una función a llamar cuando se haga y, opcionalmente, una función que sustituya a la inicialización normal de la escucha. Este último se utiliza, por ejemplo, cuando hay que proporcionar comandos momentáneos o personalizados para ciertas funciones.

La clase va tomando las órdenes de la cola según llegan, dando suficiente tiempo entre frase y frase para que acabe de pronunciarse. Este tiempo se ha calculado empíricamente con los ajustes por defecto del lector de pantalla Chromevox. Cuando una frase debe ser enunciada, se hace de dos formas:

1. Si se detecta un lector de pantalla activo (ahora mismo la aplicación solo detecta Chromevox), se vuelca la frase en un elemento HTML oculto que tiene el atributo `aria-live` con valor `assertive`. Este atributo causa que cualquier cambio en el elemento sea leído por el lector de pantalla. Así logramos hacer que el lector lea lo que queremos, sin entrar en conflictos de uso de Web Speech API (World Wide Web Consortium, 2019).

2. Si no se detecta lector alguno, se utiliza la Web Speech API (World Wide Web Consortium, 2019) directamente, concretamente la función `speechSynthesis.speak`.

La clase permite conectar una función a la que llamar cuando se recoja lo que ha dicho un usuario. Esta función recibe, si la confianza es suficiente (mayor que un 75%), la transcripción. Después, se analiza esta transcripción para saber si encaja con algún comando. Los comandos se definen en la constante `SVGVoiceGrammars`, que solo es accesible por la función `voiceParse(sentence)`. Esta función se encarga de recibir una transcripción e iterar por los comandos hasta encontrar aquel con el que encaje.

La definición de un comando consta de tres partes: un nombre que lo identifica, una expresión regular y la información a extraer. En el Código 6.5 podemos ver un ejemplo de una definición de un comando (en este caso, el comando que mueve el mapa). La expresión regular se compara con la transcripción para obtener el comando que ha sido pronunciado por el usuario. La información a extraer es la que se devuelve a la función que pide el procesamiento de la transcripción. Se debe indicar el nombre de la información y su posición en la expresión regular, para después extraerla y pasarla a la función correspondiente. Por ejemplo, el Código 6.5, citado anteriormente, extrae la dirección a la que se quiere mover el mapa, con el nombre “direction”.

Código 6.5: Definición del comando que permite mover el mapa

```
{
    name: 'move',
    pattern: /^(mover) (mapa |mapas |plano )?(a |hacia |para )?(la derecha|la izquierda
        ↪ |arriba|abajo)/i,
    extract: [
        { name: 'direction', position: 4 }
    ]
}
```

Cuando una transcripción encaja con un comando, se extrae la información que el comando indica y se devuelve como un objeto, que tiene siempre una propiedad `name` que contiene el nombre del comando, y tiene tantas propiedades como datos se quisieran extraer. Por ejemplo, el procesamiento de "mover mapa a la derecha" tendría como resultado el Código 6.6. Este resultado es ya un objeto que se puede tratar para saber la acción a realizar.

Código 6.6: Resultado de procesar un comando de voz

```
{
    name: 'move',
    direction: 'la derecha'
}
```

Con definiciones como la del Código 6.5 se conforma el repertorio de comandos disponibles en la aplicación, que puede observarse en la Tabla 6.2.

6.3.1.5 Búsqueda

El modo de exploración permite realizar búsquedas utilizando el nombre del edificio que se quiera encontrar. El componente *front-end* se encarga de recoger la información necesaria para que el *back-end* realice la búsqueda. En este caso, basta con el nombre del edificio. Se devolverá como resultado cualquier lugar cuyo nombre contenga la cadena de caracteres escrita por el usuario, de forma no sensible a mayúsculas o minúsculas. Por ejemplo, buscar "aul" devolverá "Aulario I", "Aulario II", "Aulario III" y "Facultad de Ciencias IV: Aulario" como resultado.

Un formulario recoge la cadena de caracteres que el usuario escribe. La cadena se envía al servidor utilizando uno de los URL disponibles (Tabla 6.1), en concreto `/map/data/s/name/n`, donde n es la cadena de caracteres. El servidor utiliza la información de accesibilidad almacenada para ejecutar la

Comando (expresión regular)	Función y ejemplo
(mover) (mapa | mapas | plano)?(a |hacia |para)?(la derecha|la izquierda|arriba|abajo)/	Mueve la visualización del mapa en la dirección indicada "Mover mapa hacia la derecha"
(alejar|acercar)(mapa)?	Acerca o aleja la visualización del mapa "Acercar mapa"
(buscar) (w+|d+)+	Realiza una búsqueda del nombre que se diga "Buscar Aulario 2"
(seleccionar |elegir |escoger |ver)(número |resultado)*(w+)+	Selecciona uno de los resultados de búsqueda "Seleccionar número tres"
((acceder a)|(acceso a)|(ir a)) (((cálculo de)|(calcular)|(calculo de))+)*(ruta)	Lleva al usuario al modo de navegación "Ir a cálculo de rutas"
((ir)|(calcular ruta))*desde (w|d)+) hasta ((w|d|)+)	Calcula y comienza la simulación de una ruta que vaya desde el primer punto hasta el segundo "Ir desde Aulario 2 hasta Rectorado"
(repetir|volver|decir) (paso|a decir)	Cuando la guía de navegación se haya mostrado, este comando pronuncia el paso seleccionado "Repetir paso"
(decir paso número)+ (d+)	Cuando la guía de navegación se haya mostrado, este comando pronuncia un paso en concreto "Decir paso número tres"
siguiente paso	Cuando la guía de navegación se haya mostrado, este comando avanza al paso siguiente "Siguiente paso"
paso anterior	Cuando la guía de navegación se haya mostrado, este comando retrocede al paso anterior "Paso anterior"
(hacia d(ó|o)nde estoy mirando)	La aplicación anuncia la orientación del usuario "Hacia dónde estoy mirando"
(apagar|dejar de escuchar)	Detiene el control por voz "Dejar de escuchar"

Tabla 6.2: Comandos disponibles para el control por voz

búsqueda. En concreto, busca entre la propiedad `name` definida en la tabla `property`.

El resultado está en formato JSON, y tiene el aspecto que tiene el Código 6.7. Para cada edificio cuyo nombre contiene la cadena, devuelve su nombre, su identificador y sus coordenadas en EPSG:25830 (MapTiler Team, 2017), para que pueda después centrarse el mapa en dicho resultado, si es finalmente seleccionado.

Código 6.7: Resultado de la búsqueda de la cadena "aul"

```
{
    "code":200,
    "results":[{
        "id":16,
        "name":"AULARIO GENERAL I",
        "centerx":716955.590873918,
        "centery":-4251233.24499736
    },
    {
        "id":22,
        "name":"AULARIO GENERAL II",
        "centerx":717444.93870502,
        "centery":-4251399.25399798
    }]
}
```

Los resultados se muestran en una lista que aparece después de realizar la búsqueda. Esta lista tiene un botón para ocultarla, y es enfocada en cuanto los resultados están listos. Esto facilita la navegación y la lectura de los resultados por parte de un lector de pantalla. Junto a cada resultado, un botón aparece para centrar el mapa en él.

6.3.2 Modo de navegación

El modo de navegación se ha implementado de forma que utiliza caminos prefijados para encontrar la mejor ruta por ellos desde un punto inicial hasta otro final. El algoritmo es el siguiente:

1. Se toma como entrada el identificador del edificio de partida y el de destino[9].

[9] El *front-end* es el encargado de, a través de la información que el usuario introduzca (el nombre del edificio), obtener el identificador de cada edificio.

2. Para cada edificio, encuentra un acceso desde el que partir y al que llegar, respectivamente. Si se ha especificado una discapacidad, obtendrá accesos compatibles con dicha discapacidad.

3. Una vez se tienen los accesos, se trasponen a la capa de Caminos buscando el vértice más cercano a ellos. Si los caminos se han introducido correctamente, ese vértice estará justo encima de la entrada.

4. Sabiendo de qué vértice a qué vértice debe calcularse la ruta, se aplica el algoritmo de Dijkstra con la implementación que proporciona PostGIS.

   Si un camino tiene obstáculos que afectan a la discapacidad indicada, se elimina y se busca otra forma de llegar.

   Si un camino tiene ayudas para la discapacidad indicada, se le da prioridad máxima.

5. En el caso de que no se obtengan accesos compatibles con la discapacidad o que no pueda calcularse una ruta que evite obstáculos relacionados con ella, se aplica el algoritmo de nuevo pero sin tener en cuenta la discapacidad. En el *front-end*, se alerta al usuario de esta situación.

6. El resultado de aplicar el algoritmo se devuelve como JSON, que contiene cada uno de los segmentos de la ruta. Los segmentos unen cruces de caminos con otros cruces.

En el cliente, la ruta se procesa para mejorar su comprensión. En primer lugar, se buscan puntos de interés para cada cruce. Esto se hace pidiendo al servidor una lista de edificios cercanos a dicho cruce, para después comprobar cuáles están situados a la derecha o a la izquierda del mismo. De esta forma, más tarde en la ruta final podemos indicar al usuario qué tiene a sus lados para que pueda orientarse mejor.

En segundo lugar, se calcula para cada cruce, cuál va a ser el giro que el usuario va a tener que hacer. Para ello, se calcula un vector que comienza en la posición del usuario y termina en el siguiente cruce. Después, se calcula el ángulo de ese vector respecto al norte y, teniendo en cuenta los giros que haya hecho el usuario anteriormente, se obtiene el siguiente giro. Este proceso se explica en profundidad en la Apartado 6.3.2.1.

Por último, convertimos los cruces en pasos de la ruta. Todos los cruces que consistan en seguir recto y que tengan el mismo punto de interés relacionado son combinados en un mismo paso de la ruta. El resto forman cada uno un paso propio, en el que se indica qué puntos de interés tiene cercanos, cuál es la siguiente acción a tomar y durante cuanto tiempo (o distancia). Por ejemplo: “Tienes Aulario 2 a la derecha. Gira a la izquierda y camina 15 pasos”.

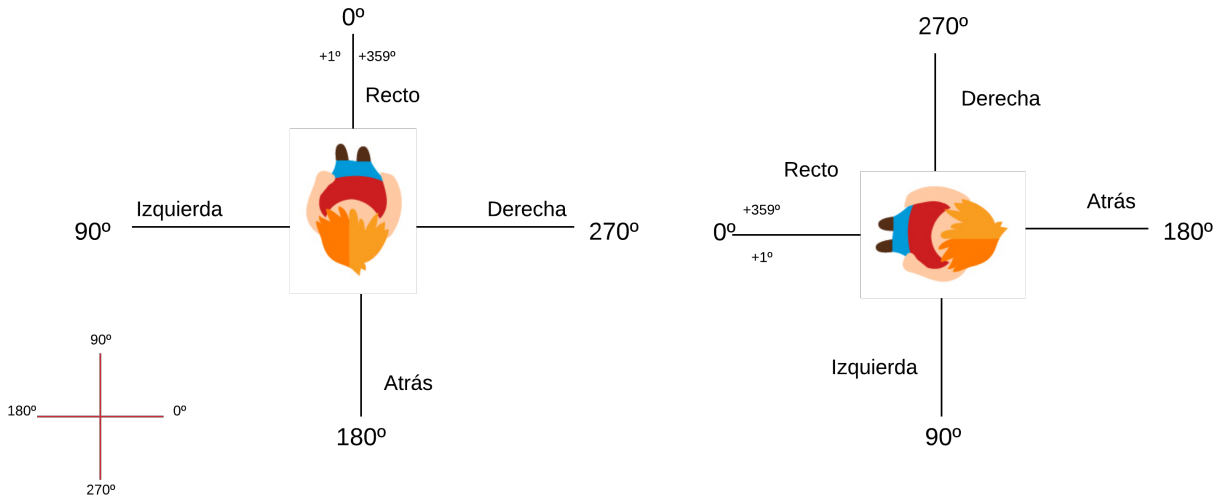


Figura 6.6: Sistema egocéntrico utilizado en dos posiciones, junto a una referencia de la representación habitual de los ángulos (abajo izquierda)[10].

6.3.2.1 Cálculo de giros

Un usuario parte, en la ruta, de un acceso de un edificio. Ese acceso tiene, respecto al norte, un ángulo hacia el que está orientado. Este ángulo es conocido en la base de datos y es desde el que partimos para el resto de cálculos.

Como el cálculo de los giros se realizan en el cliente, se utiliza el sistema de referencia del visor y los ángulos que JavaScript calcula en él. Este sistema es el mismo que aparece en la esquina izquierda de la Figura 6.6. Sin embargo, los giros se expresan de forma egocéntrica, ya que así lo prefieren los usuarios (Kalia, Legge, Roy & Ogale, 2010). Esto significa que debemos transformar los ángulos obtenidos a nuestro sistema, que puede ser apreciado visualmente en la Figura 6.6.

La ecuación que, desde un ángulo respecto al Norte α, obtiene la orientación egocéntrica, es la siguiente:

$$P(P_A, P_B) = \alpha(P_A, P_B) - D_m - 90$$

Siendo

$P(P_A, P_B)$ Función que devuelve el ángulo egocéntrico.

$\alpha(P_A, P_B)$ Ángulo respecto al Norte entre los dos puntos.

D_m Desvío por proyección del mapa. Este valor computa la diferencia entre lo que entendemos

[10]Contiene gráficos creados por *Freepik.com* (https://www.freepik.com/free-photos-vectors/people)

subjetivamente como “ir recto” y el ángulo real que supone respecto al Norte. Empíricamente, y en la proyección utilizada, son 17°.

-90 Efectúa la transposición de un sistema en el que los 0° corresponden al lugar en el que, en el sistema original, se encontraban los 90°. Esto se aprecia mejor comparando, en la Figura 6.6, la referencia original con la del sistema utilizado.

Esta ecuación nos devuelve el ángulo ajustado a nuestro sistema egocéntrico. Ahora, debemos conocer qué ángulo tiene el usuario en él. Para tener en cuenta los giros que el usuario haya dado anteriormente, aplicamos:

$$PA(P_A, P_B) = P(P_A, P_B) + D_g$$

Siendo

$PA(P_A, P_B)$ Función que devuelve el ángulo egocéntrico ajustado a la orientación del usuario.

$P(P_A, P_B)$ Ángulo en el sistema egocéntrico.

D_g Desvío acumulado por los giros que ha hecho el usuario (0 si mira recto, 90 si mira hacia la izquierda, -90 si mira hacia la derecha[11]). Por ejemplo, para un usuario que inicialmente caminaba recto, gira a la izquierda y luego a la derecha, adopta valor cero).

Esta última ecuación devuelve un valor en grados que podemos utilizar para clasificarlo en un giro concreto. Esto se realiza por rangos, como puede observarse en la Figura 6.7.

6.4 Despliegue

Como se ha indicado en la metodología, el sistema se despliega utilizando la capa gratuita de AWS. Este servicio proporciona acceso a servidores virtuales donde podemos ejecutar tanto nuestra aplicación basada en NodeJS como la base de datos PostgreSQL (y el *plug-in* PostGIS). A continuación, en la Figura 6.8, podemos observar una ilustración de la arquitectura planteada para este sistema.

Una parte muy importante de una aplicación cualquiera es su seguridad. En nuestro caso, no existe transmisión ni almacenamiento de datos personales del usuario al servidor, por lo que podemos obviar su tratamiento. Sin embargo, si que consideramos la seguridad desde el punto de vista de la Interconexión de Redes. En primer lugar, controlamos las conexiones que pueden realizarse a todos los componentes de la arquitectura mediante los Grupos de Seguridad que AWS nos permite configurar.

[11] Los conceptos de izquierda y derecha son relativos y en este caso se toman respecto a la proyección normal del mapa

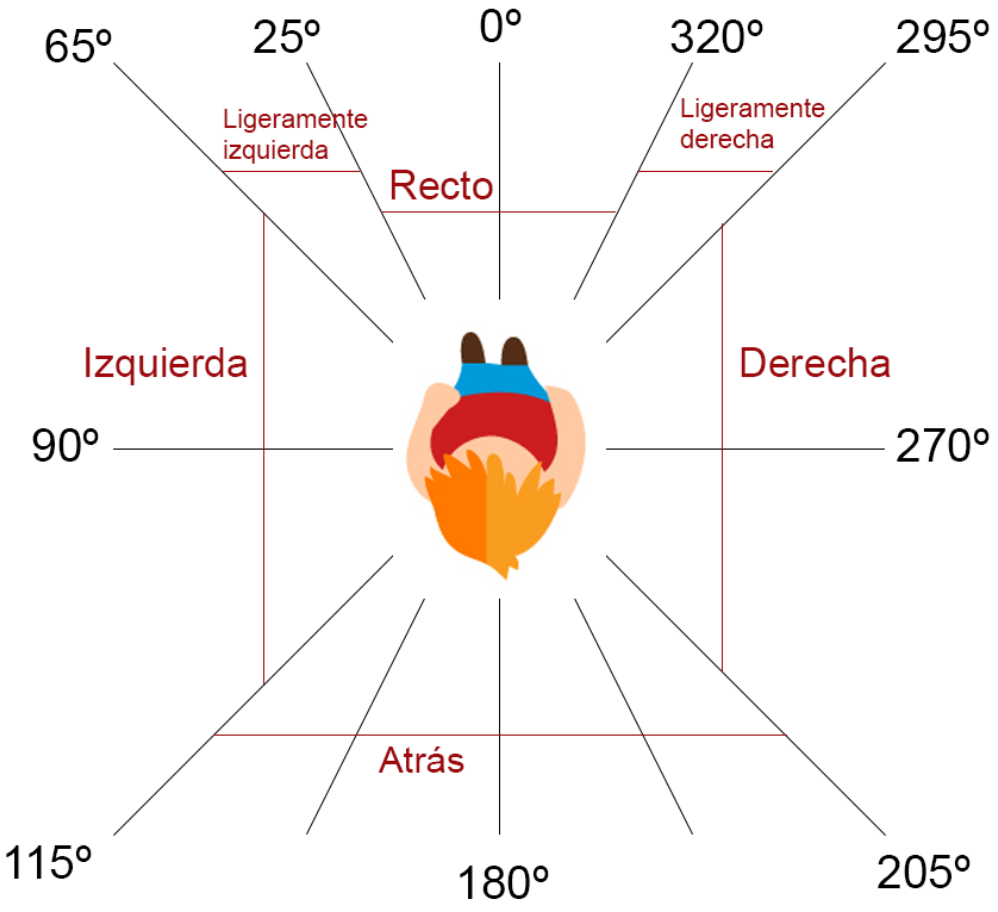


Figura 6.7: Rangos que transforman los ángulos egocéntricos obtenidos en direcciones

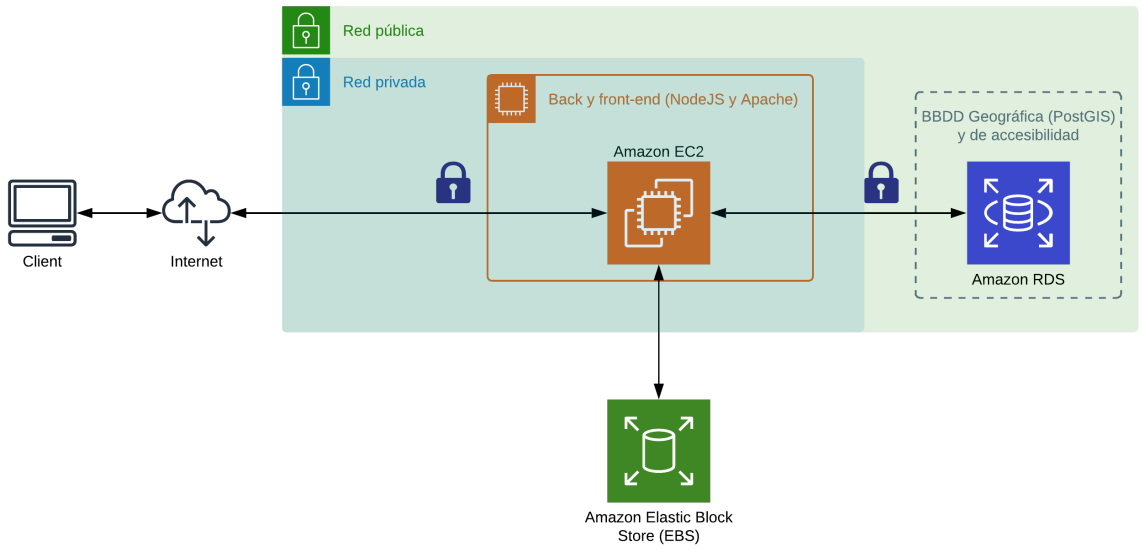


Figura 6.8: Esquema de la arquitectura del sistema una vez desplegado en AWS

Debemos diferenciar entre las ACL y los Grupos de Seguridad que ofrece AWS para el control del tráfico. Estos últimos se aplican a instancias[12] individuales, mientras que las listas se aplican a todas las instancias que pertenezcan a esa red. Otra diferencia importante es que los Grupos de Seguridad tienen estado, es decir, se permitirá tráfico de salida si se trata de una respuesta a un tráfico cuya entrada se permitió, ignorando así las reglas establecidas de salida. En la arquitectura actual, los dos servidores se encuentran bajo la misma instancia, por lo que crear una lista o un grupo tendría el mismo efecto, si no fuera por este hecho.

AWS organiza la red virtual en subredes. En nuestro caso, existen tres subredes: dos para la base de datos[13] y una donde se encuentran los servidores.

Sabiendo que el servidor Apache escucha en los puertos 80 y 443 y, el servidor NodeJS, en el 3000, la configuración actual del grupo de seguridad de la instancia de servidor es:

- Se permite la entrada de peticiones HTTP (80) y HTTTPS (443) a la instancia desde cualquier origen.
- Se permite la entrada de conexiones SSH (22) a la instancia desde cualquier origen
- Se rechaza el resto de conexiones, incluyendo aquellas al puerto 3000. Con esto conseguimos que el servidor NodeJS sólo sea accesible localmente, es decir, desde el otro servidor.

Por otro lado, la ACL que se aplica a las subredes solo permite tráfico entrante de tipo TCP (lo que

[12] Una instancia equivale a una máquina virtual que existe en la nube de Amazon en vez de en el equipo local
[13] Es un requisito impuesto por Amazon que una base de datos esté conectada en dos subredes con zonas geográficas distintas

incluye HTTP y SSH). La base de datos sólo tiene un puerto en funcionamiento, el 5432, que es el utilizado por PostgreSQL, y su Grupo de Seguridad permite todas las conexiones. Esto se debe a que, obligatoriamente y por diseño de Amazon, la base de datos debe accederse mediante Internet. Por lo tanto, debemos permitir la conexión desde cualquier lugar.

En segundo lugar, consideramos la seguridad en la administración. Los servidores y la base de datos se administran remotamente, uno mediante SSH y otro mediante cualquier cliente de PostgreSQL. Los primeros utilizan un par de claves pública y privada, mientras que la segunda utiliza unas credenciales usuario/contraseña. Esta configuración viene definida por Amazon y se confía en ella para la seguridad en la administración. La contraseña de la base de datos la generamos con una utilidad para que tenga la suficiente aleatoriedad y fuerza. La parte privada del par de claves de la instancia nunca abandona el equipo local de desarrollo. Además, la conexión a la base de datos se realiza mediante SSL. Esto ocurre así de forma obligatoria, definiendo un parámetro de configuración de AWS (`rds.force_ssl`) que fuerza a la base de datos a utilizar SSL. El certificado utilizado en esta conexión es proporcionado directamente por Amazon. Para prevenir errores de confianza, importamos el certificado a la instancia del servidor de la forma que se observa en los comandos del Código 6.8.

Código 6.8: Comandos que importan el certificado de Amazon (rds.pem)

```
1 openssl x509 -in rds.pem -inform PEM -out rds.crt
2 sudo mkdir /usr/share/ca-certificates/extra
3 sudo cp rds.crt /usr/share/ca-certificates/extra/rds.crt
4 sudo dpkg-reconfigure ca-certificates
5 sudo update-ca-certificates
```

6.4.1 Ajustes

Los ajustes son una parte importante de una aplicación y se convierten en especialmente relevantes cuando se trata de mejorar la accesibilidad. En este apartado vamos a comentar como se ha afrontado la gestión de la configuración del usuario.

La pantalla de Ajustes presenta dos partes diferenciadas: los controles que permiten cambiar la configuración y una previsualización de la misma. La previsualización refleja los cambios estéticos que se realizan al mapa a través de los ajustes y va cambiando conforme se modifican los valores de los controles. Al final de la pantalla, aparece un botón que permite guardar los ajustes. Se guardan utilizando *cookies*, porque son compatibles con todos los navegadores y podrían, en un futuro, enviarse al servidor fácilmente si fuera necesario. Los aspectos que el usuario puede modificar son:

- Tipo de mapa:
  - Completo: además de la información geográfica de nuestra base de datos, muestra una capa obtenida de OpenStreetMap[14]. Esto permite una aplicación más visual en la que es más fácil orientarse.
  - Reducido: solamente muestra el dibujo de los edificios almacenados como información geográfica en nuestra base de datos. Muy útil para personas con discapacidad cognitiva, que consideren que la versión completa contiene mucha información, o simplemente para el usuario que prefiera una vista más limpia.
- Ajustes generales:
  - Habilitar animaciones.
  - Fuente del texto.
  - Tamaño del texto.
  - Longitud del paso (en metros).
  - Velocidad andando.
- Versión textual del mapa:
  - Radio de búsqueda de lugares cercanos.
- Versión visual del mapa:
  - Fuente del texto en el mapa.
  - Color de los edificios.
  - Color del borde de los edificios.
  - Color del fondo.
- Marcadores del mapa:
  - Color del texto.
  - Color de fondo del texto.
  - Opacidad del color de fondo.
- Rutas:
  - Color de la línea de ruta.

[14]https://www.openstreetmap.org

  - Color de punto resaltado de la ruta.

- Ubicación:
  - Color del círculo de ubicación.
  - Tamaño del punto de ubicación.

Cuando el botón de guardar se pulsa, se recorren todos los controles, guardando su valor en una *cookie* con el mismo nombre que dicho control. Así, estos ajustes son accesibles desde cualquier parte de la aplicación y puede modificarse su comportamiento según ellos.

6.5 Accesibilidad en mapas en línea

La accesibilidad de un mapa web se logra combinando un conjunto de estándares y herramientas de las WCAG y de la especificación ARIA. Es importante remarcar que el mapa es implementado utilizando SVG como base, porque es la única tecnología web de gráficos que soporta accesibilidad decente. Esto se debe a que SVG es un lenguaje de marcado que representa elementos como XML, así que se pueden incluir atributos de accesibilidad.

6.5.1 Aplicación de las pautas de accesibilidad

En el Código 6.9, puede observarse un extracto del código que representa un edificio en nuestro mapa. Algunos atributos irrelevantes en términos de accesibilidad han sido eliminados por brevedad. El atributo `role` se utiliza para declarar que el contenido de la forma es una imagen (rol `img`) y que tienen un significando dependiendo de su apariencia (rol `graphics-symbol`). Después, los atributos ARIA proveen una etiqueta y una descripción. Una etiqueta es un texto corto que nombra un elemento. La descripción, sin embargo, es un texto más largo que proporciona más información sobre el elemento. En el código, combinamos al atributo ARIA `describedby` con el element SVG `desc` referenciando uno desde el otro. De esta forma, la información está disponible tanto para PA como para motores de búsqueda.

Código 6.9: Renderización de un elemento (en este caso un edificio) con información de accesibilidad

```
<a id="feature-link-1"
   tabindex="0"
   role="graphics-symbol img"
   aria-label="Building name"
   aria-describedby="feature-link-1-desc" ...>
```

```
    <desc id="feature-link-1-desc">
        Building name and description
    </desc>
    <path ...></path>
</a>
```

Utilizamos este conocimiento para implementar atributos ARIA al código SVG generado por un visor de Leaflet[15]. En primer lugar, el visor recibe la información geográfica del servidor. Entonces, renderiza un documento SVG. Hacemos el código accesible de dos formas: proveemos un renderizador personalizado que envuelve en un enlace cada forma que representa un edificio. Esto permite al usuario navegarlos utilizando el teclado. Una vez el edificio recibe el foco, la segunda estrategia entra en acción: la información que describe el edificio se carga del servidor a atributos ARIA. Utilizando la combinación correcta de atributos, permitimos al lector de pantalla enunciar correctamente información útil para el usuario discapacitado. Esta secuencia está ilustrada en la Figura 6.9.

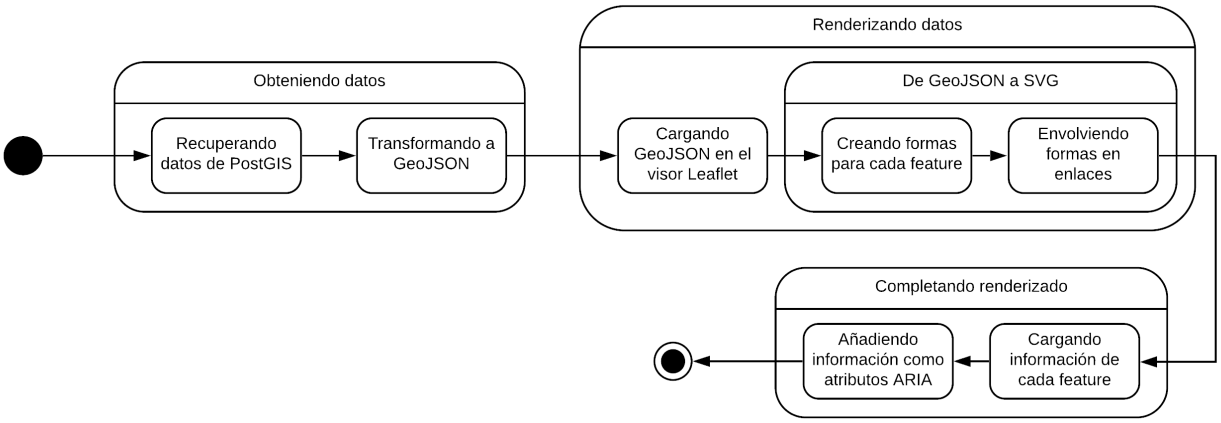


Figura 6.9: Secuencia utilizada para renderizar datos geográficos con información de accesibilidad

Adicionalmente, se expone información sobre el mapa completo a los PA. Esto significa que no solo los edificios o los marcadores tienen información de accesibilidad individualmente, sino que el mapa en sí también la tiene. De esta forma, un usuario puede saber de un vistazo lo que puede esperar del mapa. Por ejemplo, el Código 6.10 muestra cómo nuestra aplicación indica que el mapa muestra información geográfica de la Universidad de Alicante. La estrategia para hacer estos datos legibles a un PA es la misma que antes: atributos ARIA relacionados con los elementos `title` y `desc`. Sin embargo, en este caso no envolvemos nada en un enlace. Como estamos trabajando ahora con un elemento HTML, el atributo `tabindex` será suficiente para permitir que el mapa reciba el foco utilizando el teclado. Solo

[15]https://leafletjs.com

debemos envolver elementos SVG para permitir su interacción.

Código 6.10: Información de accesibilidad en el mapa completo

```
<svg
   tabindex="0"
   id="MAIN_SVG"
   aria-labelledby="svgrendertitle"
   aria-describedby="svgrenderdesc"
   role="graphics-document document" ...>
   <title id="svgrendertitle">
      Accessible map of the University of Alicante
   </title>
   <desc id="svgrenderdesc">
      Accessible map of the University of Alicante, which belongs to the TFG
      by Sergio Juan Armero, supervised by Sergio Luján Mora.
   </desc>
   <g id="rootGroup"> ... </g>
</svg>
```

A lo largo de la memoria, se ha explicado que algunos datos se muestran como listas en la interfaz de usuario (por ejemplo, los lugares cercanos). Para que estas listas sean accesibles, hay que pensar en que cada elemento de las mismas debe ser accesible por teclado y por ratón. Es sencillo hacer que esto ocurra encerrando el contenido de cada elemento en un enlace (elemento HTML `<a>`). De esta forma, se acceden tanto por teclado como con ratón y los lectores de pantalla leen correctamente su contenido.

Sin embargo, no todas las listas que mostramos pretenden que sus ítems tengan una acción al pulsar sobre ellas. Por ejemplo, los resultados de búsqueda tienen la interacción en un botón contenido en cada elemento. Por lo tanto, no admiten encerrar el contenido en un enlace, puesto que estaríamos rompiendo con el concepto del comportamiento percibido (Apartado 3). En este caso, hacemos que la lista obtenga el foco añadiendo `tabindex="0"`. Así, el lector puede leerla entera. La navegación por las distintas opciones vamos a recrearla mediante los botones, que si admiten inoteracción. Si el contenido de los botones depende de su contexto visual para comprenderse, utilizamos `aria-label` para añadir la información de contexto que falte, para personas invidentes.

Por otra parte, los botones que aparecen en la aplicación contienen dos atributos, que son: `title` y `aria-label`. El atributo `title` nos permite dar más información sobre la función del botón que

aparecerá visualmente en un recuadro cerca de la posición del cursor, cuando este se sitúe sobre el botón. Lo utilizamos cuando es necesario dar más información de la que visualmente recibe un usuario sin discapacidad. El atributo `aria-label` proporciona información sobre la función del botón a los lectores de pantalla, así que su contenido sería leído cuando el usuario seleccione el botón. De esta forma, podemos cubrir necesidades distintas para cada situación, ya que la información que necesita, por ejemplo, una persona con capacidad visual no es la misma que la que necesita una con discapacidad visual.

Sobre los enlaces que puedan aparecer en la aplicación (elementos `<a>`), hemos descartado por completo el atributo `title` para darles información de accesibilidad (Ball, 2013). Optamos por utilizar textos de ancla significativos e independientes de su contexto, de forma que cuando un lector de pantalla lo lea en voz alta, un usuario pueda entenderlo correctamente. En el Código 6.11 podemos ver un ejemplo de un enlace en su forma preferible, y otro que utiliza el atributo `title`. Vemos también que redactamos un texto de ancla fácilmente comprensible, para concretar su función y hacerlo más breve y, por lo tanto, mejor para usuarios con discapacidad cognitiva.

Código 6.11: Enlace sin y con atributo “title”

```
<!-- Do -->
<a href="/route">Calcular rutas</a>

<!-- Don't -->
<a href="/route" title="Accede al cálculo de rutas">Rutas</a>
```

Una funcionalidad que también aporta a la accesibilidad es el hecho de que podemos determinar el orden en el que se recorren los edificios cuando se utiliza el teclado para tabular entre ellos. Aunque ahora mismo solo se permite un orden (de oeste a este), la aplicación está preparada para admitir en un futuro otra ordenación. Para conseguir esto, proporcionamos una interfaz llamada `ITabOrder` con dos métodos, uno que obtiene su nombre y otro que obtiene el orden de la tabulación.

El orden de tabulación se expresa mediante un array[16] (`number[]`) que enumera los identificadores de los edificios por orden de tabulación. Por ejemplo, el array `[ 3, 2, 1 ]` hace que, cuando se comience a recorrer el mapa con el teclado, se visite en primer lugar el edificio con identificador 3, en segundo lugar aquel con identificador 2 y, en último lugar, aquel con identificador 1. Creando una clase que implemente la interfaz `ITabOrder` podemos hacer que los edificios se recorran en cualquier orden deseado.

[16]Un array o arreglo es una colección de datos de un mismo tipo

7 Resultados

A continuación, se expondrán los resultados del desarrollo anteriormente descrito, centrándonos en la interfaz de usuario y su accesibilidad.

7.1 Interfaz de usuario

Como se puede observar en la Figura 7.1, la aplicación tiene dos áreas principales. Una barra lateral a la izquierda contiene botones de control para mover y acercar o alejar el mapa. Después, tres botones son mostrados: controles de voz, localización y orientación. El primero activa el reconocimiento de voz, el segundo mueve el mapa a la ubicación del usuario y el tercero enuncia la orientación del usuario (hacia qué edificio está mirando).

En el mapa, los edificios tienen marcadores que indican su nombre. Son agrupados en algunos niveles de *zoom*, con la intención de evitar la sobrecarga de información (como observamos en la Figura 7.2). El número de marcadores agrupados en cada uno es mostrado, y sus colores cambian dependiendo de este número. Cuando el usuario navega con el teclado, lo hacen a través de los grupos de marcadores en vez de los propios marcadores. Si el *zoom* se aumenta a un nivel más cercano, los marcadores normales aparecen, y el usuario navega directamente a través de los edificios.

En la Figura 7.1, también podemos observar una sección que contiene una lista de edificios cercanos, llamada "Cerca de ti". Esto tiene el objetivo de funcionar como una alternativa textual a la representación gráfica del mapa. Junto al nombre de cada edificio, aparece un enlace para comenzar la navegación a ese lugar. Tras esta sección, otra llamada "Elementos a la vista" lista edificios que se encuentran en el campo de visión del mapa.

Los usuarios pueden adaptar la alternativa textual a sus necesidades, ajustando cómo de lejos el mapa buscará edificios cerca de ellos. Un control se muestra justo al principio de la sección. Este control indica la distancia con la que el mapa está trabajando y algunos botones para modificarla. Si un usuario la modifica, por ejemplo aumentándola, la lista de edificios cercanos es recalculada, como puede observarse en la Figura 7.3

La barra lateral puede ocultarse a voluntad. Como se puede ver en la parte superior izquierda de la

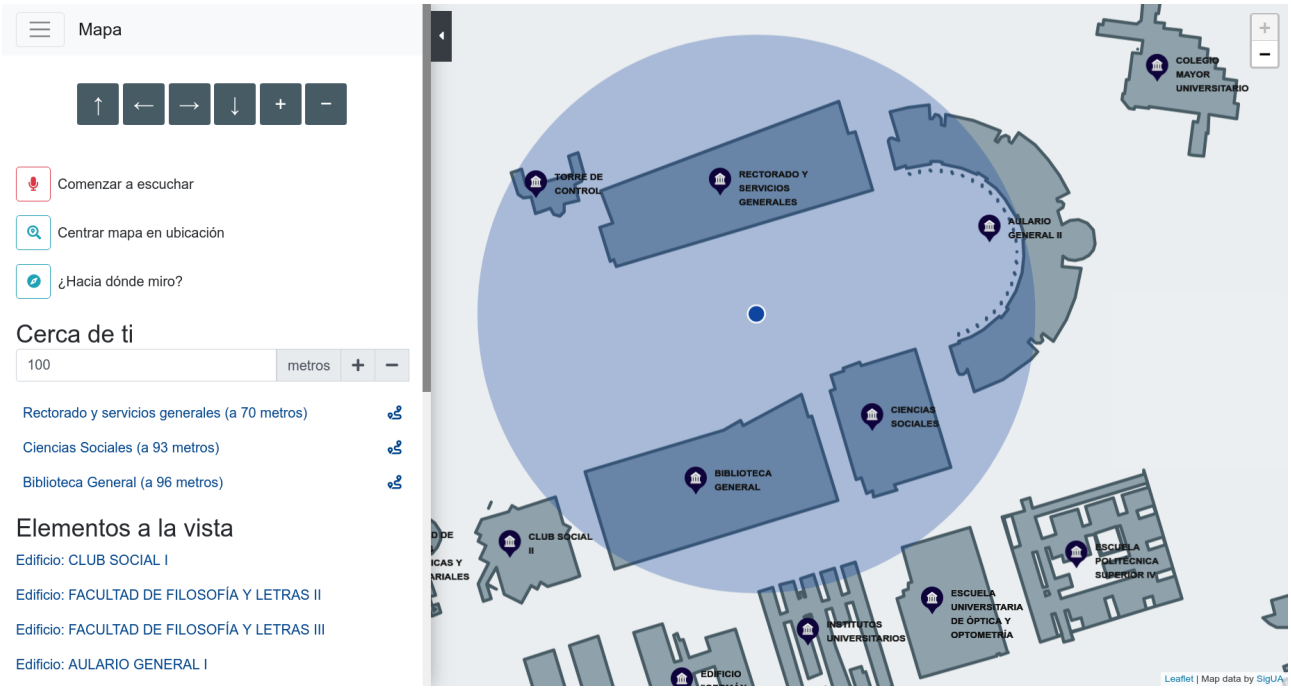


Figura 7.1: Página principal de la aplicación

Figura 7.1, la interfaz de usuario también contiene un menú de "hamburguesa" con enlaces al modo de navegación y a la página de ajustes.

7.2 Accesibilidad de la interfaz de usuario

Respecto a los grupos de marcadores mencionados previamente, son etiquetados con el nombre del edificio con la mayor prioridad. Esta prioridad se establece manualmente por los gestores de datos para cada uno de los edificios. Esto nos permite diferenciar entre edificios más o menos importantes. En caso de empate, el grupo se llama igual que el edificio más cercano a la posición del grupo de marcadores.

Además del mapa web, se ofrece una representación textual. Esto es una versión alternativa de mostrar los mismos datos, y es crucial para la accesibilidad. Utiliza la ubicación del usuario en el momento para mostrar una lista de sitios cerca de él o ella.

Sin embargo, no todos los usuarios discapacitados lo son visualmente. Para dar soporte a las personas discapacitadas física o intelectualmente, el mapa proporciona varias formas de interaccionar con él. Puede ser explorado y controlado con el teclado, con el ratón o haciendo clic en unos botones de control que se hallan en pantalla.

También puede ser utilizado por voz. Los controles por voz funcionan para el modo de exploración y para el de navegación. En el modo de exploración, los usuarios pueden usar su voz para buscar y

Figura 7.2: Marcadores agrupados en el mapa, debido a un zoom lo bastante alejado como para que los marcadores contenidos en la vista pudieran sobrecargar la información. Los colores cambian según cuántos marcadores haya agrupados

Figura 7.3: Comparación entre dos distancias diferentes en la sección "Cerca de ti"

seleccionar edificios, mover y cambiar el *zoom* del mapa, escuchar su ubicación o mover el mapa a su ubicación. Estos controles se han implementado utilizando la Web Speech API (World Wide Web Consortium, 2019). Esta API permite al desarrollador acceder a dos funciones: síntesis y reconocimiento de voz. Nuestro proyecto utiliza las dos para comunicarse con el usuario por voz.

En primer lugar, utilizamos el reconocimiento de voz para escuchar lo que el usuario dice. Cuando el usuario dice un comando en voz alta, la API proporciona a nuestro sitio web con una transcripción de lo que ha dicho. Entonces, utilizando expresiones regulares, discernimos si la transcripción encaja con alguno de los comandos disponibles. Si así es, la acción correspondiente es ejecutada.

En segundo lugar, utilizamos la síntesis de voz para responder al usuario cuando ha usado un comando de voz. Por ejemplo, si el usuario realiza una búsqueda por voz, los resultados son enunciados en voz alta utilizando esta API. Sin embargo, esto solamente pasa cuando un lector de pantalla no está activo. Si lo está (actualmente solo puede detectarse Chromevox) la responsabilidad de hablar recaerá en el lector. Esto se hace con el objetivo de evitar conflictos, ya que el PA debe prevalecer sobre nuestro propio uso de la síntesis de voz.

En el modo de navegación, la voz puede usarse para buscar rutas entre dos edificios. Una vez la ruta es simulada, los usuarios pueden navegar por la guía (los pasos que deben seguir para llegar a su destino) con comandos de voz. Esto es muy útil para una navegación manos-libres, ya que el usuario puede concentrarse en sus alrededores y en seguir la ruta.

En tercer lugar, también se proporciona una representación alternativa. Las instrucciones se muestran tanto en una lista numerada como dibujadas en el mapa, como puede verse en la Figura 7.4. En el mapa se muestran como líneas y puntos de giro, que se iluminan cuando se selecciona el paso correspondiente en la guía. La lista numerada muestra cada paso necesario para llegar desde el paso inicial hasta el final. Estos pasos se indican con la misma estructura sintáctica, lo que ayuda a usuarios

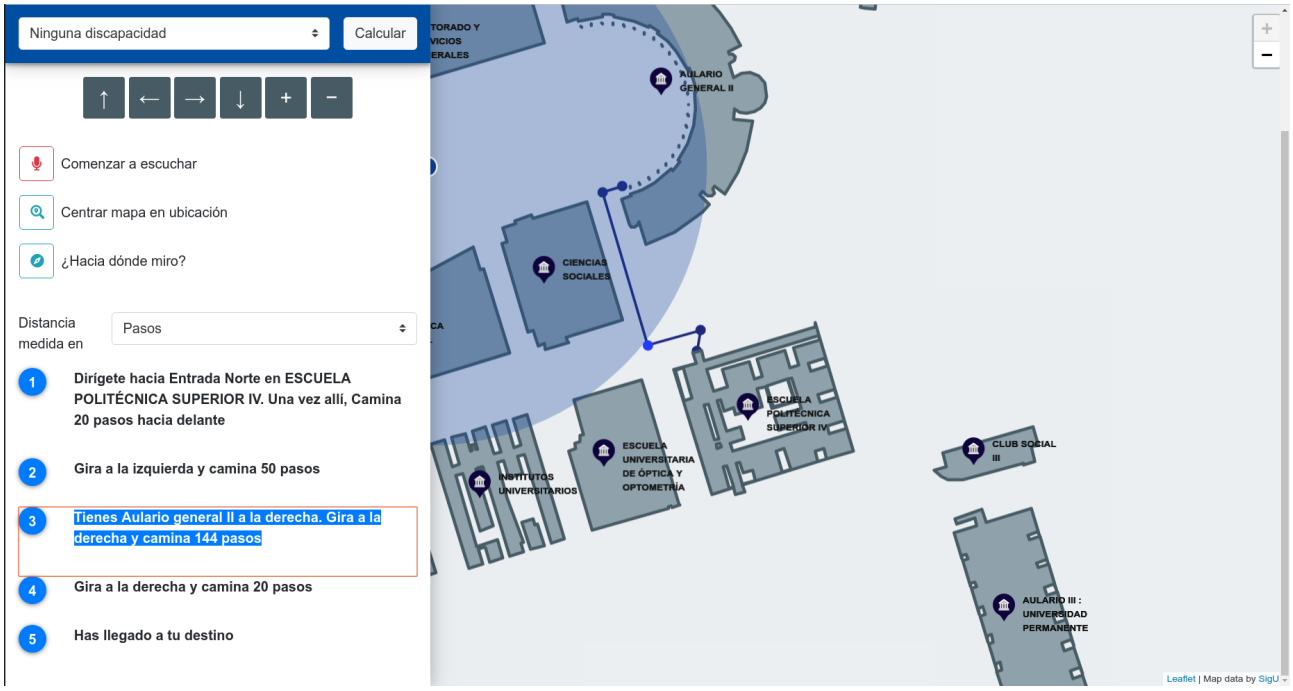


Figura 7.4: Modo de navegación con una ruta simulada para un usuario con discapacidad visual

con discapacidad intelectual. Este modo también tiene en cuenta la accesibilidad de la ruta calculada. Esto significa que la aplicación intentará primero obtener una ruta que no tenga obstáculos y, si eso no es posible, se mostrará una ruta con obstáculos, así como una alerta sobre esta situación para que el usuario sea consciente. En ambos casos, los caminos que tienen ayudas para personas con discapacidad tienen más prioridad en el cálculo de la ruta. Los obstáculos dependen del tipo de discapacidad que el usuario presente y solo pueden ser detectados si éste la indica, usando el selector visible en la esquina superior izquierda de la Figura 7.4. Este tipo de navegación, que tiene en cuenta la naturaleza de los caminos, está siendo implementada actualmente en Google Maps. Aunque no está disponible en todo el mundo, la última versión de este servicio es capaz de simular rutas que no cuenten con obstáculos arquitectónicos (Akasaka, 2018).

Otra forma de lograr la accesibilidad es la personalización. Nuestro proyecto viene con un gran abanico de ajustes. En el mapa web, el color de los edificios, de los indicadores de orientación y localización, de las líneas de ruta y de los puntos de giro se puede cambiar. Esto es crucial para personas daltónicas y con otro tipo de discapacidad visual, porque pueden ajustar los colores a sus necesidades. El tamaño del texto, así como el tipo de fuente, pueden ser cambiados también. Esto ayuda a personas con baja visión.

Una de las opciones que ofrecen los ajustes es cambiar el tipo de mapa, entre “reducido” y “completo”. En el primer tipo, los edificios almacenados en la base de datos se dibujan encima de un fondo monocromo, que por defecto es gris claro y que el usuario puede cambiar en Ajustes. Es una

Figura 7.5: A la izquierda, una zona del mapa en versión reducida. A la derecha, la misma zona en versión completa

vista reducida del mapa que ofrece una visión simplificada de la geografía de la zona, lo que ayuda a usuarios con discapacidad cognitiva, ya que hay menos información concentrada. El segundo tipo muestra una capa obtenida de OpenStreetMap sobre la que se dibujan los edificios. La capa de OpenStreetMap se obtiene pasando al visor Leaflet el URL donde este servicio aloja los mapas (`https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png`, donde s, z, x, e y son valores completados por Leaflet). En la Figura 7.5 podemos observar las diferencias entre los dos tipos.

Como el modo de navegación provee las distancias en varias unidades, el tamaño de cada paso y la velocidad andando puede ser personalizada también en el perfil del usuario, para que la aplicación calcule instrucciones más precisas. En concreto, el modo navegación ofrece tres unidades distintas: metros, pasos y tiempo caminando. Son las mismas unidades que se utilizaron en otra investigación (Kalia et al., 2010) donde observamos que dieron un buen resultado. En él, los comentarios de los usuarios dieron las siguientes conclusiones, que hemos seguido en nuestro proyecto:

- Se utilizan tanto indicaciones egocéntricas como indicaciones utilizando norte, sur, este y oeste. Al final del artículo concluyen que los usuarios prefieren las primeras, así que son las que nosotros utilizamos.

- Aparte de las rutas, la aplicación tiene un modo de exploración que permite al usuario conocer lo que tiene a su alrededor.

- La distancia de las rutas se miden en metros, número de pasos y segundos andando. En nuestro caso, cuando los segundos sobrepasan el minuto, se expresan en minutos y segundos.

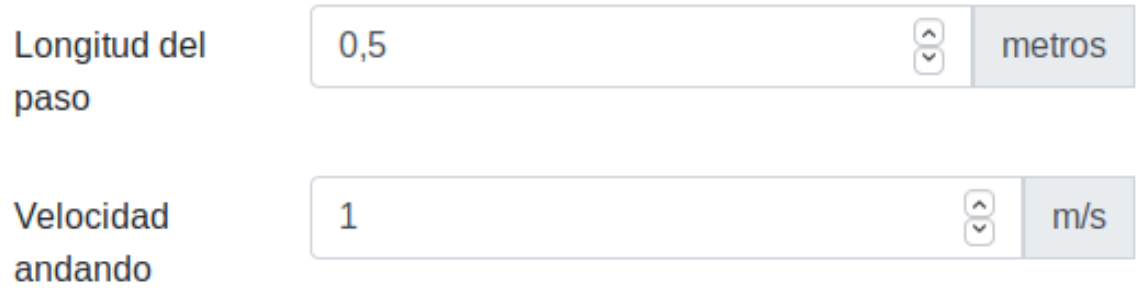


Figura 7.6: Extracto de la pantalla de Ajustes donde se configuran la distancia de paso y la velocidad andando

- Las instrucciones son escuchadas de una en una por los usuarios, que navegan por ellas moviéndose por una lista.
- La primera instrucción siempre indica la posición y orientación del usuario. El resto indica la distancia a caminar y la dirección de giro.
- Las instrucciones se hacen siempre con el mismo formato. Es decir, el orden de los sintagmas es el mismo. Esto ayuda al aprendizaje de uso de la aplicación.

El estudio también muestra que las personas con buena visión caminan 1.85 pies por paso y 2.92 pies por segundo. Aquellas con baja visión, 1.98 pies por paso y 3.43 pies por segundo. La variación de la longitud del paso en un individuo es muy pequeña, según su estudio también. Esto justifica el uso del paso como unidad de distancia y nos da valores por defecto que añadir a nuestro proyecto: 0.5 metros cada paso y una velocidad de 1 metro por segundo. Podemos observar un extracto de la pantalla de Ajustes donde aparecen estas opciones de personalización en la Figura 7.6.

La aplicación ofrece una previsualización de los ajustes seleccionados, como puede observarse en la parte derecha de la Figura 7.7. De esta forma, los usuarios pueden explorar cómodamente múltiples combinaciones de colores y tamaños de texto y, una vez están seguros con su elección, guardar esos ajustes. Los ajustes son almacenados en *cookies*, que son recuperadas cada vez que el usuario accede al mapa. La presentación es entonces modificada dependiendo en el valor de éstas. Esta tecnología fue elegida en vez de otras como el *Local Storage* porque las *cookies* son compatibles con todos los navegadores y pueden ser transferidas al servidor. Ahora mismo eso no es necesario, pero podría serlo en futuras ampliaciones del proyecto.

La aplicación también puede informar la usuario sobre su ubicación y su orientación. Esto significa que los usuarios discapacitados visual e intelectualmente (que normalmente tienen los mayores problemas de orientación) pueden saber en todo momento dónde están y hacia dónde miran. El nombre

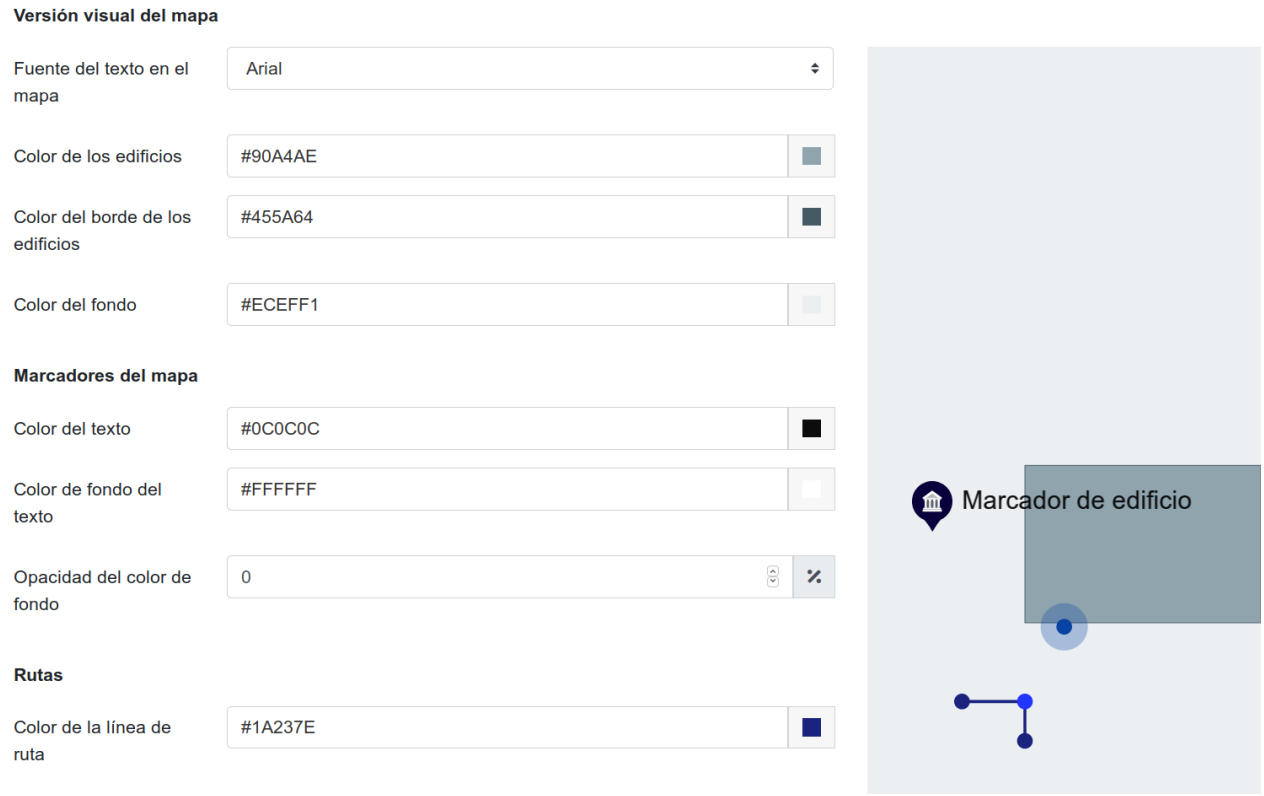


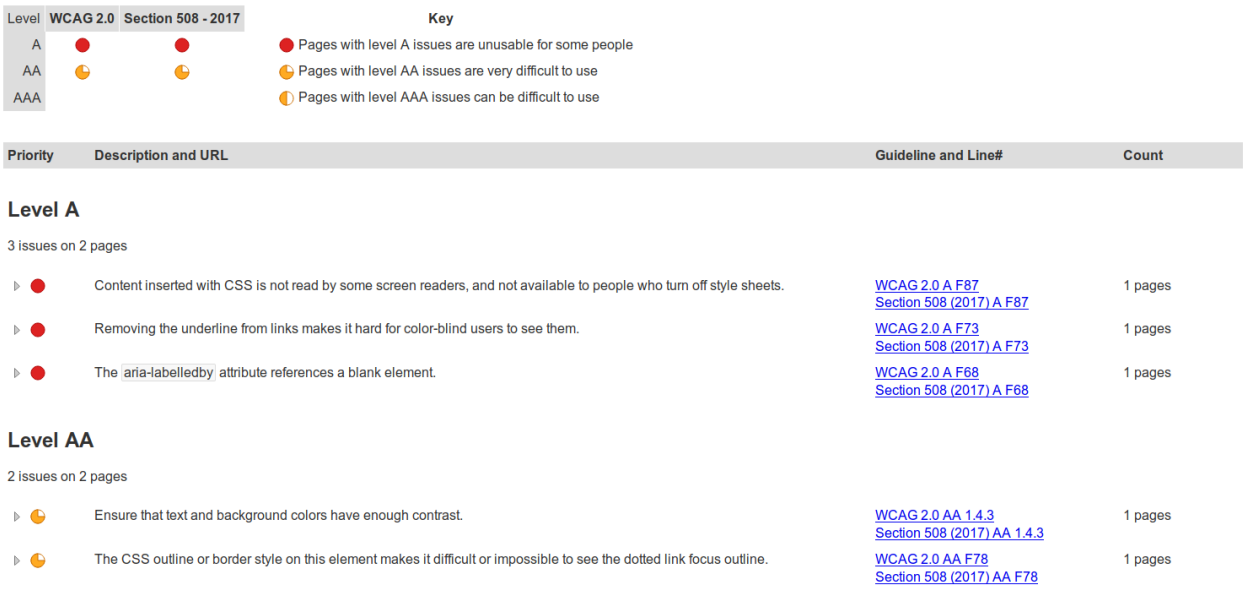
Figura 7.7: Previsualización de los ajustes

del edificio al que están orientados se lee en voz alta y es mostrado también en pantalla. Este usuario puede accionar este comportamiento mediante un botón. Además, durante la navegación, se indica información sobre los alrededores. Esto solo ocurre cuando el usuario navega por los distintos pasos de la guía. De esta forma, el usuario puede saber si está siguiendo las instrucciones de forma correcta y tener una experiencia más segura.

7.3 Evaluación de la accesibilidad

La aplicación web desarrollada pasó con éxito tanto el validador oficial de HTML como el de CSS. Más allá, un test automatizado de accesibilidad fue llevado a cabo, que también se aprobó. Este test, realizado con la herramienta PowerMapper[1], dio un nivel de conformidad WCAG 2.0 AAA (World Wide Web Consortium, 2008). La versión WCAG 2.1 (World Wide Web Consortium, 2018b) no pudo probarse porque no existen aún herramientas para esa versión en el momento del test. Estos resultados deberían ser comprobados en un test humano. En la Figura 7.8 podemos ver las únicas alertas que arrojó la prueba, originados en código del *framework* Bootstrap y del *plug-in* Leaflet, por lo que no dependen de nosotros. El código desarrollado por nosotros pasa las pruebas de accesibilidad. Hay

[1]https://www.powermapper.com/



Level	WCAG 2.0	Section 508 - 2017	Key
A			Pages with level A issues are unusable for some people
AA			Pages with level AA issues are very difficult to use
AAA			Pages with level AAA issues can be difficult to use

Priority	Description and URL	Guideline and Line#	Count
Level A			
3 issues on 2 pages			
	Content inserted with CSS is not read by some screen readers, and not available to people who turn off style sheets.	WCAG 2.0 A F87 Section 508 (2017) A F87	1 pages
	Removing the underline from links makes it hard for color-blind users to see them.	WCAG 2.0 A F73 Section 508 (2017) A F73	1 pages
	The `aria-labelledby` attribute references a blank element.	WCAG 2.0 A F68 Section 508 (2017) A F68	1 pages
Level AA			
2 issues on 2 pages			
	Ensure that text and background colors have enough contrast.	WCAG 2.0 AA 1.4.3 Section 508 (2017) AA 1.4.3	1 pages
	The CSS outline or border style on this element makes it difficult or impossible to see the dotted link focus outline.	WCAG 2.0 AA F78 Section 508 (2017) AA F78	1 pages

Figura 7.8: Resultado del test automático de PowerMapper

una alerta sobre el contraste de un texto, que hemos identificado como un error de la herramienta. Pensamos que esto es así porque el texto del que habla es de color blanco sobre fondo negro, contraste más que suficiente. El resto de avisos, como decimos, vienen de código ajeno.

También se llevaron a cabo tests manuales sobre los colores, pensando en personas daltónicas. Utilizamos la herramienta de comprobación de color de WebAIM[2] y una extensión del navegador ("Let's get color blind"[3]) para simular daltonismo. Los colores por defecto cumplen con la ratio de contraste de colores de WCAG AAA incluso para esta discapacidad.

[2]https://webaim.org/resources/contrastchecker
[3]https://chrome.google.com/webstore/detail/lets-get-color-blind/bkdgdianpkfahpkmphgehigalpighjck

8 Conclusiones y trabajo futuro

El resultado obtenido muestra que es posible implementar un mapa web accesible con buena funcionalidad y un diseño moderno. Sin embargo, se requiere tener acceso a una base de datos geográfica completa con la información a representar en el mapa. Además, es esencial proveer más información, además de la geográfica, para conseguir una presentación accesible de la misma. Esto hace nuestro modelo incompatible con las APIs actuales de mapas, como Google Maps. Estas APIs normalmente cargan el mapa en un *canvas*, que como ha sido discutido anteriormente, no es candidato a ser accesible. Así que, aunque hemos implementado un mapa web accesible, se limita a aquellos que pueden permitirse mantener y alimentar una base de datos completa.

Nuestra investigación también muestra la importancia de SVG en la accesibilidad web. Cuando una imagen tiene múltiples partes que aportan significado, esta tecnología es la única forma de proveer una forma accesible de proporcionar esa información. Aunque los elementos `<img>` con texto alternativo pueden mostrar una imagen de forma accesible, debemos usar SVG si queremos características más avanzadas, como interacción o modularidad. Esto significa que el desarrollo de gráficos escalares vectoriales es fundamental para la accesibilidad web, así como la correcta implementación de la especificación por parte de todos los navegadores modernos. En este caso concreto, esto es de gran importancia, porque los mapas web pueden ayudar a personas con discapacidad con uno de los temas más problemáticos para ellos: el movimiento independiente. Mejorar la accesibilidad de mapas web permite ayudarles a convertirse en personas más autónomas y autosuficientes. Adicionalmente, nuestra solución es compatible y funciona correctamente en dispositivos móviles, así que puede ser usada para que personas discapacitadas se muevan en sus alrededores.

La accesibilidad está lejos de ser un estándar en las aplicaciones actuales. A pesar de los beneficios que trae consigo, no hay mucha concienciación sobre este entre desarrolladores y estudiantes. Las instituciones tampoco lo están poniendo fácil. Aunque las WCAG han existido casi desde la fundación de la W3C, la especificación ARIA fecha de 2014. Eso no es mucho tiempo, hablando en términos informáticos. Ahora mismo, la especificación funciona bien en casos básicos, pero se empieza a complicar cuando intentamos hacerla funcionar en aplicaciones más específicas, como mapas web. Solo para que los lectores de pantalla lean la información correctamente, hemos tenido que llevar a cabo un

estudio completo para observar cómo los lectores más usados manejan los atributos ARIA y, entonces, escoger los mejores. El panorama actual recuerda a los primeros días de la web, donde cada navegador implementaba las especificaciones a su manera. Esto es perjudicial para todos los esfuerzos por concienciar. En resumen, necesitamos estándares, especificaciones e implementaciones de accesibilidad consistentes.

Nuestra aplicación puede ser mejorada de varias formas. Primero, se podría realizar una prueba con usuarios reales, para proporcionarnos una mejor visión de la accesibilidad y usabilidad del mapa. Debemos considerar los resultados del test humano y los comentarios de los usuarios. Este *feedback* nos permitirá entender cómo la aplicación encaja con las necesidades de las personas con discapacidad. Así, podríamos elegir entre distintos algoritmos o formas de realizar distintos cálculos (por ejemplo, podríamos ponderar por distancia el voto de cada línea del proceso explicado en el Apartado 6.3.1.4.2).

Además, ahora mismo sólo se soporta la navegación en exteriores. Pensamos que una característica muy útil sería la navegación en interiores, para que los usuarios puedan trasladarse de forma segura en espacios interiores. El modo de exploración funcionaría como el de exteriores. El modo de navegación simularía rutas dentro del edificio, de forma similar a la que otros investigadores han propuesto, donde probó ser útil (Calle-Jimenez & Luján-Mora, 2016). Técnicas de posicionamiento en interiores descritas en otros estudios, que usan Bluetooth y RSSI (Cabrera-Goyes & Ordóñez-Camacho, 2018) o tags NFC (Tao, Ding, Wang & Ganz, 2017), pueden ser también adoptados para orientar al usuario.

El algoritmo utilizado para calcular las rutas utiliza caminos guardados estáticamente en la base de datos. Esto hace el mantenimiento más complicado, porque requiere esfuerzo humano en manejar esos caminos. Un algoritmo que genere caminos automáticamente utilizando la información geográfica sería beneficioso en este aspecto.

Respecto a los controles por voz, solo funcionan en Chrome en este momento, porque es el único navegador que soporta la Web Speech API. En cuanto otros navegadores la implementen, deberíamos comprobar que funcione correctamente en nuestra aplicación. También se debería revisar los comandos para que sean más inteligentes, consiguiendo que los usuarios puedan hablar de una forma más natural.

Los usuarios también deberían poder obtener información sobre sus alrededores a su voluntad en el modo de navegación. Con al implementación actual, sólo es posible hacer eso en el modo de exploración. También queda pendiente de añadir una opción para cambiar el orden de tabulación de los edificios a otros, además del actual orden de oeste a este.

Como se comentaba en la introducción, existen varios tipos de discapacidad que afectan de forma distinta. La aplicación debería ofrecer características que ayuden a personas con discapacidades como la discalculia (Butterworth et al., 2011), por ejemplo con una sintaxis muy simplificada, algo parecido al concepto de “Simple English” (Kaiser, 2013).

Bibliografía

Accessibility of Google Maps API, Google Issue Tracker. (2017). Recuperado desde https://issuetracker.google.com/issues/69541792

Ahmetovic, D., Gleason, C., Ruan, C., Kitani, K., Takagi, H. & Asakawa, C. (2016). NavCog: a navigational cognitive assistant for the blind. En ACM (Ed.), *International Conference on Human-Computer Interaction with Mobile Devices and Services* (pp. 90-99). Florence.

Akasaka, R. (2018). Introducing "wheelchair accessible" routes in transit navigation. Recuperado desde https://www.blog.google/products/maps/introducing-wheelchair-accessible-routes-transit-navigation/

Aoyama, H. (2019). Design Accessible Maps. Recuperado desde https://phase.com/magazine/design-accessible-maps/

Ball, D. (2013). I thought title text improved accessibility. I was wrong. Recuperado desde https://silktide.com/blog/2013/i-thought-title-text-improved-accessibility-i-was-wrong

Brajnik, G. (2009). Validity and reliability of web accessible guidelines. En ACM (Ed.), *ACM SIGACCESS conference on Computers and accessibility* (pp. 131-138).

Butterworth, B., Varma, S. & Laurillard, D. (2011). Dyscalculia: From Brain to Education. *Science (New York, N.Y.)* 1049-53.

Cabrera-Goyes, E. & Ordóñez-Camacho, D. (2018). Posicionamiento en espacios interiores con Android, Bluetooth y RSSI. *Enfoque UTE, 9(1)*, 118-126.

Calle-Jimenez, T., Eguez-Sarzosa, A. & Luján-Mora, S. (2019). Design of an Architecture for Accessible Web Maps for Visually Impaired Users. En *Advances in Human Factors and Systems Interaction.* (pp. 212-232). Orlando.

Calle-Jimenez, T. & Luján-Mora, S. (2016). Accessible Online Indoor Maps for Blind and Visually Impaired Users. En ACM (Ed.), *International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 309-310). Reno, Nevada.

Campin, B., McCurdy, W., Brunet, L. & Siekierska, E. (2003). Accessible Maps for the Visually Impaired. En *SVG Open 2003*, Vancouver.

Connor, A. (2010). Perceived Affordances and Designing for Task Flow. Recuperado desde http://johnnyholland.org/2010/04/perceived-affordances-and-designing-for-task-flow/

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001). Section 24.3: Dijkstra's algorithm. En *Introduction to Algorithms* (pp. 595-601). MIT Press y McGraw–Hill.

Donoso, M. T. (2018). El pavimento podotáctil y la accesibilidad, Numbers Magazine. Recuperado desde http://numbersmagazine.com/articulo.php?tit=el-pavimento-podotactil-y-la-accesibilidad

Ducasse, J., Brock, A. & Jouffrais, C. (2017). *Accessible Interactive Maps for Visually Impaired Users.*

Ellis, K. & Kent, M. (2017). *Disability and Social Media: Global Perspectives.*

Ferraz, R. (2017). Accessibility and search engine optimization on scalable vector graphics. En *IEEE International Conference on Soft Computing & Machine Intelligence* (pp. 94-98). Port Louis.

Fisher, C. (2019). Creating Accessible SVGs. Recuperado desde https://www.deque.com/blog/creating-accessible-svgs/

Fundación Telefónica. (2018). Presentamos Mapcesible. ¡Haz visible lo accesible! Recuperado desde https://espacio.fundaciontelefonica.com/evento/presentamos-mapcesible-haz-visible-lo-accesible/

Gómez Sáez, I. (2017). GRACES WIKI. Recuperado desde https://github.com/ysmartin/graces/wiki

Instituto Municipal de Desarrollo Económico y Empleo de Córdoba. (2013). Córdoba accesible. Recuperado desde http://www.cordobaaccesible.org/index.html

International Organization for Standardization. (2014). *Guide for addressing accessibility in standards (ISO/IEC Guide 71:2014)*.

Internet Engineering Task Force. (2016). The GeoJSON Format. Recuperado desde https://tools.ietf.org/html/rfc7946.html

Kaiser, H. (2013). A close look at Simplified Technical English. Recuperado desde http://www.tcworld.info/e-magazine/technical-communication/article/a-close-look-at-simplified-technical-english/

Kalia, A. A., Legge, G. E., Roy, R. & Ogale, A. (2010). Assessment of Indoor Route-finding Technology for People with Visual Impairment. *J Vis Impair Blind*, *104*(3), 135-147.

Lewis, V. (2018). Decoding The Colors of Blindness Canes, Veroniiiica. Recuperado desde https://veroniiiica.com/2018/01/12/decoding-the-colors-of-blindness-canes/

Logan, T. (2018). Accessible Maps on the Web. Recuperado desde https://equalentry.com/accessible-maps-on-the-web/

Luján-Mora, S. (2009). Dispositivos adaptados. Recuperado desde http://accesibilidadweb.dlsi.ua.es/?menu=disp-adaptados

Luján-Mora, S. (2014). *La comunicación oral. Claves para realizar buenas presentaciones.*

MapTiler Team. (2007). EPSG:4326. Recuperado desde https://epsg.io/4326

MapTiler Team. (2017). EPSG:25830. Recuperado desde https://epsg.io/25830

Microsoft. (2017). Types of Bitmaps. Recuperado desde https://docs.microsoft.com/en-us/dotnet/framework/winforms/advanced/types-of-bitmaps?view=netframework-4.7.2

Migliorisi, H. (2016). Accessible SVGs. Recuperado desde https://css-tricks.com/accessible-svgs/

Piórkowski, A. (2011). Mysql spatial and postgis-implementations of spatial data standards. *Electronic Journal of Polish Agricultural Universities*, 1-8.

PowerMapper. (2018). WAI-ARIA Screen Reader compatibility. Recuperado desde https://www.powermapper.com/tests/screen-readers/aria/

Pressbooks Ryerson University. (2019). Types of Disabilities and Barriers. Recuperado desde https://pressbooks.library.ryerson.ca/wafd/chapter/types-of-disabilities-and-barriers/

PUNTODIS. (2017). Map's voice: la información de recorridos y puntos de interés de los planos al alcance de todos. Recuperado desde https://puntodis.com/featured_item/planos-mas-accesibles-con-maps-voice/

Singh, A., Bansal, R. & Jha, N. (2015). Open Source Software vs Proprietary Software. *International Journal of Computer Applications*, *114*(18), 26-31.

Tao, Y., Ding, L., Wang, S. & Ganz, A. (2017). PERCEPT Indoor Wayfinding for Blind and Visually Impaired Users: Navigation Instructions Algorithm and Validation Framework. En *International Conference on Information and Communication Technologies for Ageing Well and e-Health*, Porto.

Watson, L. (2014). Using the tabindex attribute. Recuperado desde https://developer.paciellogroup.com/blog/2014/08/using-the-tabindex-attribute/

Watson, L. (2018). Accessible SVG flowcharts. Recuperado desde https://tink.uk/accessible-svg-flowcharts/

WebAIM. (2016). Keyboard Accessibility. Recuperado desde https://webaim.org/techniques/keyboard/tabindex

World Health Organization. (2001). *International Classification of Functioning, Disability and Health.* World Health Organization.

World Health Organization. (2011). *World Report on Disability.* World Health Organization.

World Wide Web Consortium. (1997). World Wide Web Consortium Launches International Program Office for Web Accessibility Initiative. Recuperado desde https://www.w3.org/Press/IPO-announce

World Wide Web Consortium. (2008). Web Content Accessibility Guidelines (WCAG) 2.0. Recuperado desde https://www.w3.org/TR/WCAG20/

World Wide Web Consortium. (2011). Scalable Vector Graphics (SVG) 1.1 (Second Edition). Recuperado desde https://www.w3.org/TR/SVG11/

World Wide Web Consortium. (2015). HTML Canvas 2D Context. Recuperado desde https://www.w3.org/TR/2dcontext/

World Wide Web Consortium. (2017). Accessible Rich Internet Applications (WAI-ARIA) 1.1. Recuperado desde https://www.w3.org/TR/wai-aria-1.1/

World Wide Web Consortium. (2018a). Using ARIA. Recuperado desde https://www.w3.org/TR/using-aria/

World Wide Web Consortium. (2018b). Web Content Accessibility Guidelines (WCAG) 2.1. Recuperado desde https://www.w3.org/TR/WCAG21/

World Wide Web Consortium. (2019). Web Speech API. Recuperado desde https://w3c.github.io/speech-api/